High-Color C++ Programmable Controller with Touch

MoaTouch

User's Manual

2014-07-03

COMFILE TECHNOLOGY

"Everything for Embedded Control"

www.ComfileTech.com

Table of Contents

The MoaTouch is a high-color, C++ programmable touch controller which combines touch input, colorful output, RS-232/485 and Ethernet communication, and digital & analog I/O ports.

The MoaTouch library provides convenient access to all IO features and a rich set of user interface features such as vector graphics, raster graphics, TrueType fonts and PNG images. Courtesy of the GNU toolchain, users can take advantage of many C++11 features such as lambda functions, initializer lists, range-based for loops and more.

Features

- High-Color TFT display
- Touch Input
- RS-232 and RS-485 communication
- Non-volatile Memory
- Digital Input and Output
- Analog Input
- Real-Time Clock
- TCP/IP Ethernet
- Vector graphics (lines, rectangles, circles, curves, text, images, and many other drawing primitives)
- Solid colors and linear/radial gradients
- Alpha blending (transparency and partial transparency)
- TrueType font and Unicode Support
- PNG image file support
- MicroSD Card with FAT32 file system support
- Field upgradable firmware

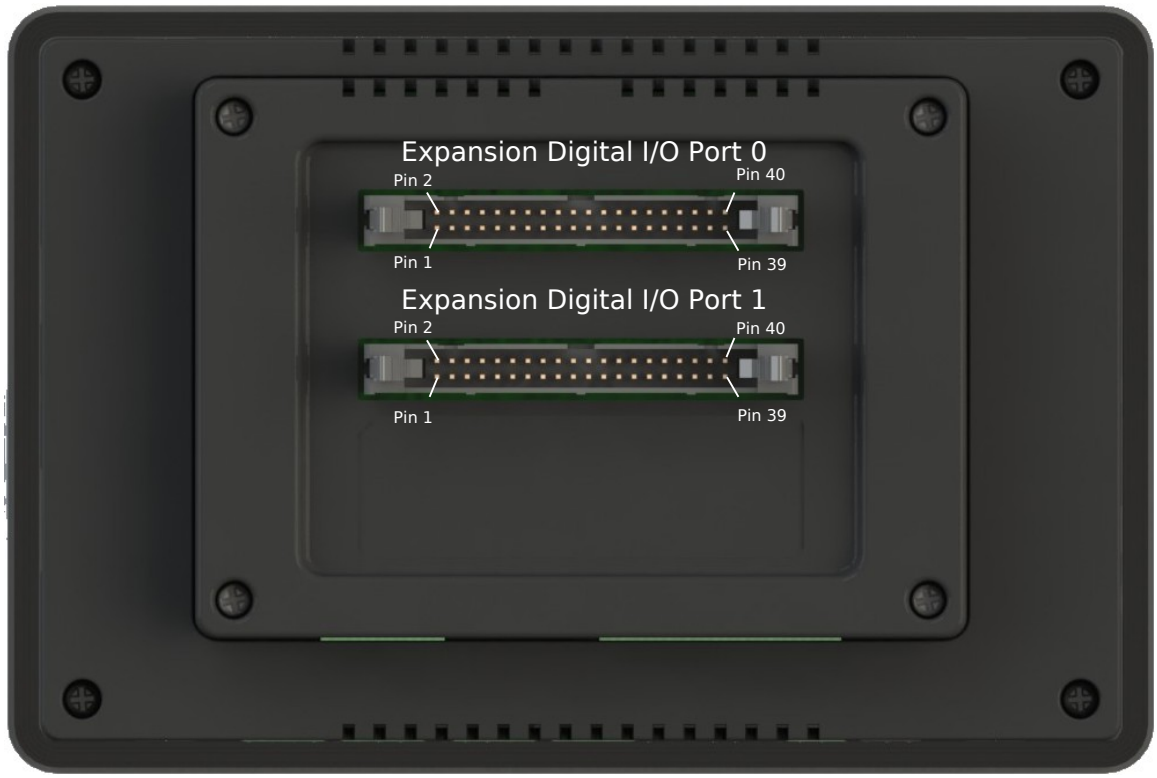
Specifications

- 16-bit color, 800×480 or 480x800 7” TFT LCD
- 4-wire Resistive Touch Screen
- 168MHz ARM Cortex-M4F MCU
- 4MB RAM
- 512 bytes of non-volatile memory
- 2 RS-232 Serial Ports
- 2 RS-485 Serial Ports
- 36 Digital Inputs w/ Interrupt
- 36 Digital Outputs
- 4 12-bit Analog Inputs (Optional)
- 1 MicroSD Card Slot
- 1 USB Console/Firmware Update Port
- 1 Ethernet Port

Physical Connections

Expansion Digital I/O Ports

Expansion Digital I/O Ports 0 and 1 (in code: `exDigitalInputs` and `exDigitalOutputs`) are connected to the host processor over an I^2C bus. The other Digital I/O port (in code: `digitalInputs` and `digitalOutputs`) are connected direction to the host processor's GPIO pins.



	Expansion Digital I/O Port 0	Expansion Digital I/O Port 1
Pin 1, 10, 19, 28, 30, 39	Ground	
Pin 29, 40	V Source Voltage	
Pin 2~9	`exDigitalInputs[0~7]`	`exDigitalInputs[16~23]`
Pin 11~18	`exDigitalInputs[8~15]`	`exDigitalInputs[24~31]`
Pin 20~27	`exDigitalOutputs[0~7]`	`exDigitalOutputs[16~23]`
Pin 31~38	`exDigitalOutputs[8~15]`	`exDigitalOutputs[24~31]`

	Input									Output										
Port 0	0	2	4	6	G	9	11	13	15	0	2	4	6	G	G	9	11	13	15	V
	G	1	3	5	7	8	10	12	14	G	1	3	5	7	V	8	10	12	14	G

	Input									Output										
Port 1	16	18	20	22	G	25	27	29	31	16	18	20	22	G	G	25	27	29	31	V
	G	17	19	21	23	24	26	28	30	G	17	19	21	23	V	24	26	28	30	G

See Electrical Specifications for more information.

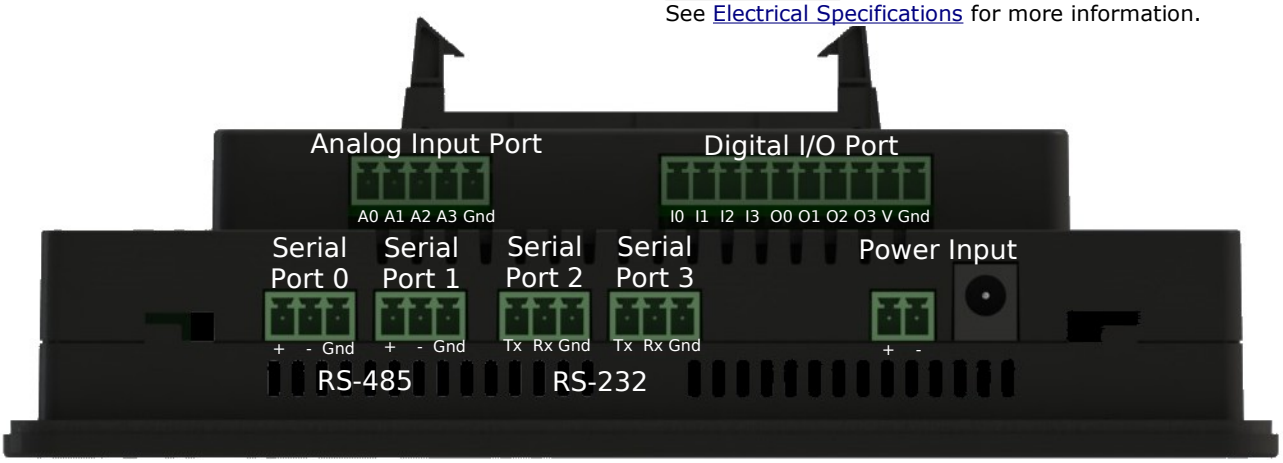
Serial, Analog and Digital I/O Ports

Analog Input Port (Optional)	
Gnd	Ground
Pin A0 ~ A3	`analogInputs[0~3]`

See Electrical Specifications for more information.

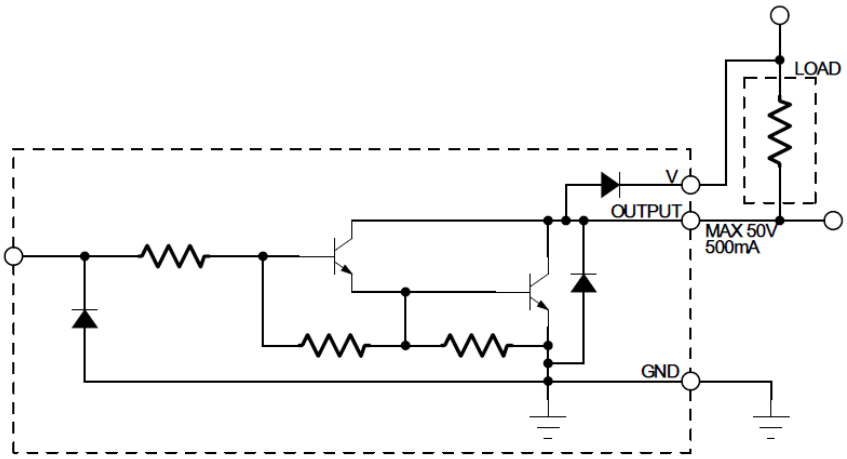
Digital I/O Port (Optional)	
Gnd	Ground
V	Source Voltage
Pin I0 ~ I3	`digitalInputs[0~3]`
Pin O0 ~ O3	`digitalOutputs[0~3]`

See Electrical Specifications for more information.



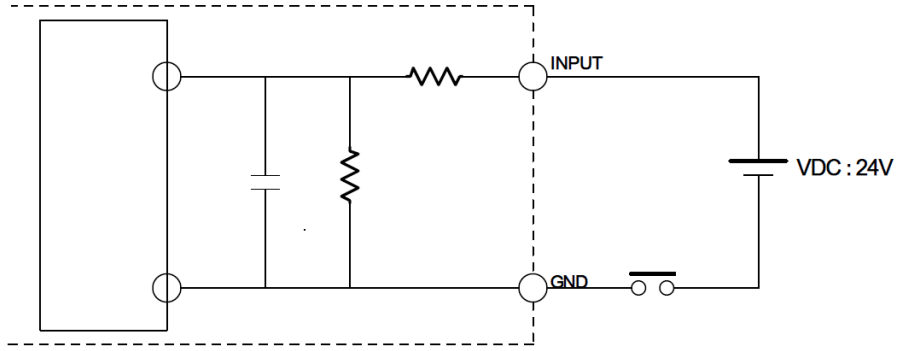
Serial Ports	
Serial Port 0 ~ 3	`serialPorts[0~3]`

Electrical Specifications



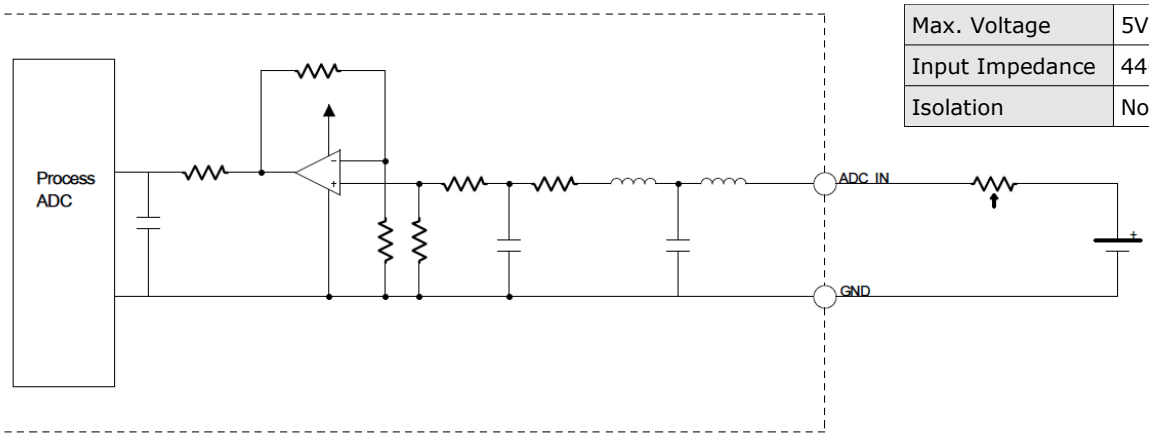
Max. V	50V
Output Current	500mA

Digital Output – Output Sink (ON creates a path to ground)



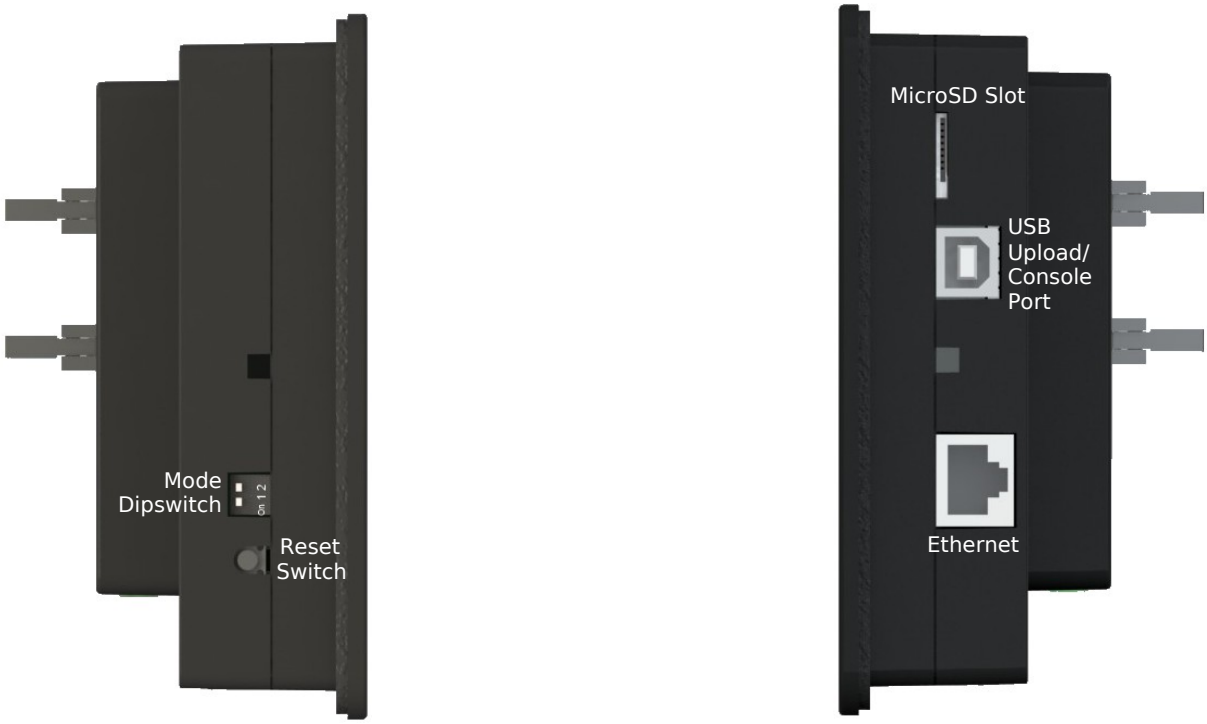
Input Voltage	24V
Input Current	2mA
Input Impedance	12KΩ
On Level	> 17V
Off Level	< 5V
Isolation	None

Digital Input



Max. Voltage	5V
Input Impedance	440KΩ
Isolation	None

Analog Input



USB Upload/Console Port

This USB port functions as either a virtual serial port (console), or a program upload port port depending on the state of the Mode Dipswitch.

Mode Dipswitch

- **Dipswitch 1 –** Reserved for future use.
- **Dipswitch 2** - Switches between DFU *Mode (On)* and *Console Mode* (Off).

- *DFU Mode* - In this mode, when the MoaTouch is first powered on, it will appear to the host PC as DFU device so programs can be uploaded to it. See Installing the MoaTouch DFU Driver for further instructions.

- *Console Mode* – In this mode, when the MoaTouch is first powered on, it will bypass DFU mode and begin executing the last program that was uploaded to it. The program can then send messages to the host PC for monitoring and debugging. See Installing MoaTouch Virtual Serial Port (Console) Driver for further instructions.

Micro-SD Card Slot

Accepts a FAT32 formatted Micro-SD Card for storing user data. Long file names are not supported which implies the following limitations:

- File names are limited to an 8 character name and a 3 character extension (a.k.a 8.3 format)
- Files created by the MoaTouch will appear in all upper-case letters despite having specified lower-case letters in code.
- Only ASCII characters are supported in file and folder names.

Ethernet

Ethernet port for connecting the MoaTouch to a network. It can be programmed using the Ethernet and Socket classes.

Reset Switch

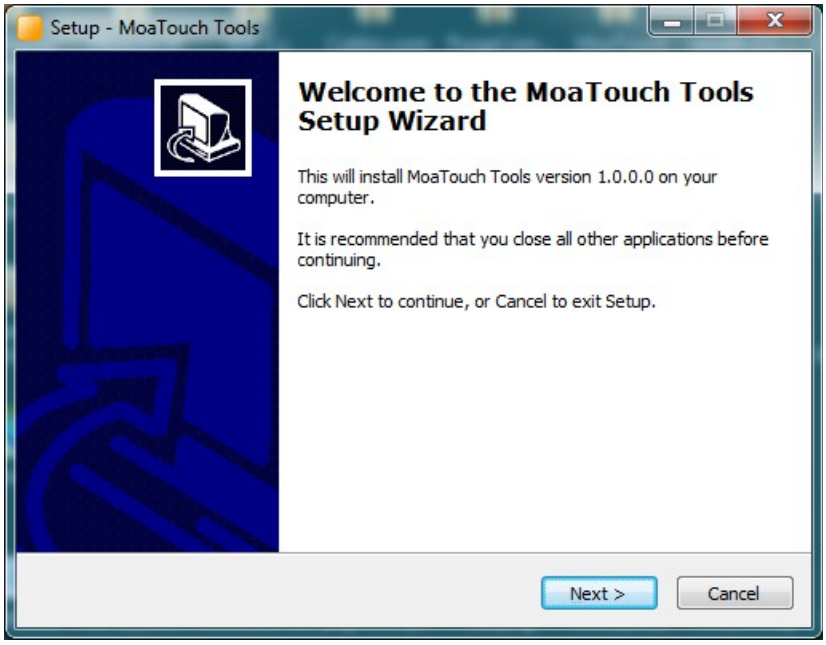
Push this switch to perform a hard reset.

Software Installation

To begin utilizing the MoaTouch a few PC tools are required

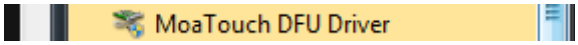
- MoaTouch Virtual Serial Port (Console) Driver
- MoaTouch Control Panel utility program
- MoaTouch Device Firmware Upgrade (DFU) Driver
- GNU ARM Embedded Toolchain

These are all distributed in a single package called "MoaTouch Tools" available for download from Comfilte Technology's website, www.ComfileTech.com. Simply run the install utility and follow the on-screen instructions to install this software on the host PC.

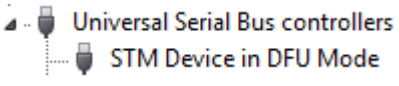


Installing the MoaTouch DFU Driver

After running the MoaTouch Tools installation utility, you'll need to install the MoaTouch DFU driver so you can upload programs the the MoaTouch. You can find a link to the driver's installation utility in Windows Start Menu, under "Comfile Tools".



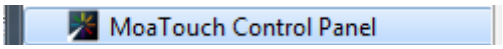
After installing the MoaTouch DFU Driver, if you set dipdwitch 2 to the ON position and power on the MoaTouch, you will see the MoaTouch appear as "STM Device in DFU Mode" in Windows Device Manager.



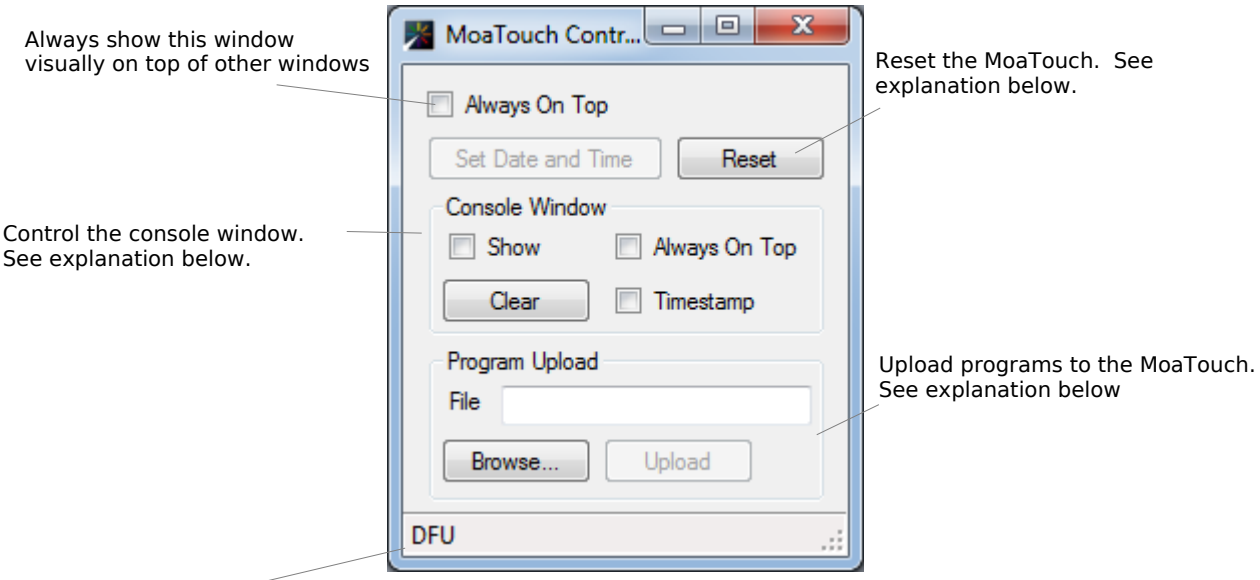
Now, using the MoaTouch Control Panel, you will be able to upload programs from the host PC to the MoaTouch.

MoaTouch Control Panel

The MoaTouch Control Panel is a program that gives the user the ability to upload programs to the MoaTouch and monitor a program's execution. Using it in concert with your software development environment will make programming the MoaTouch quite convenient. The MoaTouch Control Panel can be run from the Windows Start Menu under "Comfile Tools". This program requires the .Net Framework 4.0 (client profile) which can be downloaded from Microsoft.



When executed, the following window will appear.



Console Window:

When ta program begins executing, the MoaTouch will appear to the host PC as a virtual serial port. If the program sends a message to either standard out, or standard error, those messages will appear in the Console Window.

- "Show" – Show or hide the Console Window
- "Always on Top" – Always display the Console Window on top of all other windows
- "Clear" – Clear all text in the Console Window.
- "Timestamp" – Display a timestamp next to each line displayed in the console window. This can be useful for logging.

Program Upload:

When dipswitch 2 is in the off position and the MoaTouch is first powered on, it will appear to the host PC as a DFU device. In this mode, a program can be uploaded from to the MoaTouch. After uploading a program, the MoaTouch will immediately begin executing that program, despite the status of dipswitch 2.

- "File" – Path to the program's executable (*.hex) to be uploaded to the MoaTouch.
- "Browse..." - Use windows explorer to locate the program's executable file.
- "Upload" – Upload the program's executable to the MoaTouch, and begin executing it.

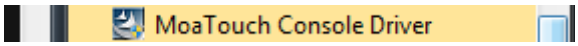
The Reset Button:

Pressing the "Reset" button will tell the MoaTouch to perform a software reset. If the MoaTouch is in DFU mode, it will exit DFU mode and attempt execution of the program. If the MoaTouch is executing a program and dipswitch 2 is in the ON position, the MoaTouch will re-enter DFU mode. If the MoaTouch is executing a program and dipswitch 2 is in the OFF position, the program will restart.

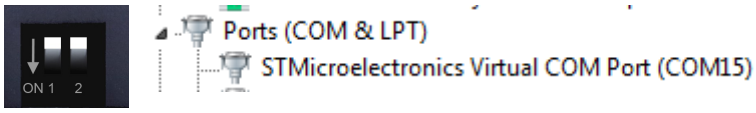
Installing MoaTouch Virtual Serial Port (Console) Driver

When dipswitch 2 is in the OFF position, the MoaTouch's USB port can function as an output console for debugging and monitoring a program's execution.

Once again, you can find a link to the driver's installation utility in Windows Start Menu, under "Comfile Tools".

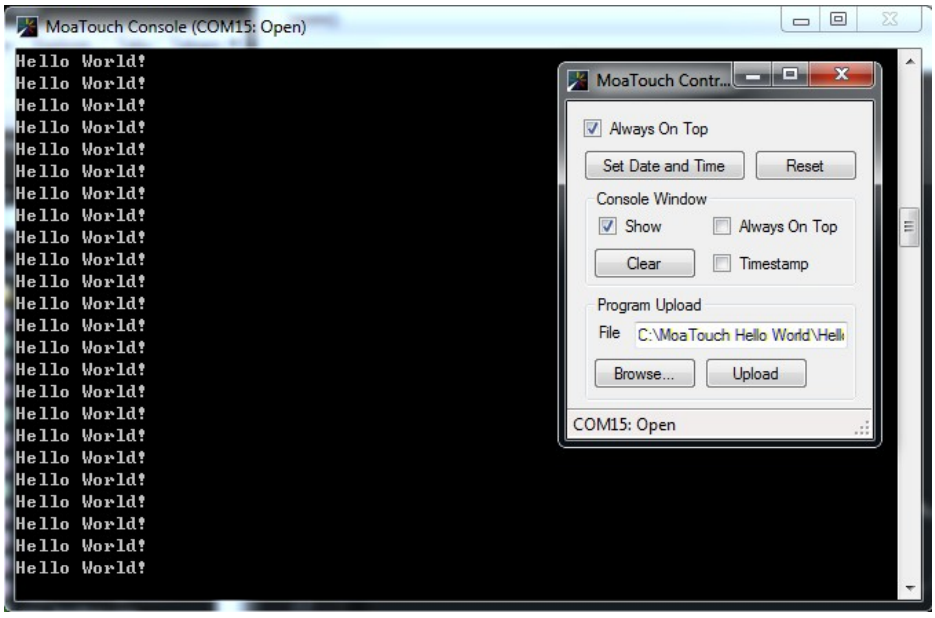


After installing the driver, if you set dipswitch 2 to the OFF position, and power on the MoaTouch, you will see the MoaTouch appear as "STMicroelectronics Virtual COM Port (COMxx)" in Windows Device Manager. The COM port number will vary.



Now, using the MoaTouch Control Panel, you can view output from the MoaTouch in the MoaTouch Control Panel's Console Window. Please be aware, that the MoaTouch will not appear as a virtual serial port until a working program is uploaded to it.

```
#include <MoaTouch.h>

using namespace MoaTouch

int main()
{
    while(true)
    {
        console.PrintLine("Hello World");
        Delay_ms(500);
    }

    return 0;
}
```



Developing Software for the MoaTouch

The MoaTouch Tools installation utility will install the GNU ARM Embedded Toolchain to the "{Installation Folder}\ComfileTools\MoaTouch Tools\GNU Tools for ARM Embedded Processors" folder. This is a variant of the GNU toolchain and includes the GCC compiler, linker, and much more. Detailed documentation on this toolchain can be found in the "share" folder. In this section, we will describe how to use this toolchain to program the MoaTouch.

The GNU ARM Embedded Toolchain can be used with many text editors, IDEs, and other tools for software development. Software developers are quite particular about their tools, so we will not advocate one specific editor or IDE. Rather we will show the general procedure one would use to compile software with this toolchain, and leave it to the reader to take this information and configure their editor, IDE, or other tools as they prefer.

We will use a simple text editor to edit the code, the Windows command console to invoke the GNU toolchain, and the MoaTouch Control Panel to upload our program to the MoaTouch. We will create a simple "Hello World!" program to illustrate this procedure.

Editing the Source Code

We create 3 files with our text editor, all in the same folder:

- HelloWorld.h – The header file for the HelloWorld class.
- HelloWorld.cpp – The implementation file for the HelloWorld class.
- main.cpp – The entry point of our program

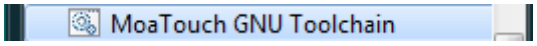
```
// HelloWorld.h
#ifndef HELLOWORLD_H
#define HELLOWORLD_H

class HelloWorld
{
public:
    void SayHello();
};

#endif
```

```
// HelloWorld.cpp
#include <cstdio>        // for printf
#include "HelloWorld.h"

void HelloWorld::SayHello()
{
    printf("Hello World!\r\n");
}
```

```
// main.cpp
#include <MoaTouch.h>
#include "HelloWorld.h"

using namespace MoaTouch;

int main()
{
    HelloWorld hw;      // create an instance of the HelloWorld class

    // loop forever
    while(true)
    {
        hw.SayHello();  // call the SayHello method to print to standard output
        Delay_ms(500);  // delay for 500ms
    }
}
```

Compiling the Source Code

In the Windows Start Menu, under "Comfile Tools" you'll find a link to "MoaTouch GNU Toolchain". This runs a batch file that adds the toolchain's bin folder to the host PC's `path` variable.



Click it, and a console window will open. Change directory (`cd`) to the folder containing the source code. Then type the following commands to compile the source code.

```
arm-none-eabi-g++ -c -mthumb -mcpu=cortex-m4 -mfpu=fpv4-sp-d16 -mfloat-abi=softfp -fsingle-
precision-constant -ffunction-sections -fdata-sections -std=c++11 -I"C:\Program Files
(x86)\ComfileTools\MoaTouch Tools\include" HelloWorld.cpp -o HelloWorld.o

arm-none-eabi-g++ -c -mthumb -mcpu=cortex-m4 -mfpu=fpv4-sp-d16 -mfloat-abi=softfp -fsingle-
precision-constant -ffunction-sections -fdata-sections -std=c++11 -I"C:\Program Files
(x86)\ComfileTools\MoaTouch Tools\include" main.cpp -o main.o
```

Explanation of each argument:

- `arm-none-eabi-g++` - The GNU C++ compiler
- `-c` – compile only, don't link
- `-mcpu=cortex-m4` – The MoaTouch's MCU is an ARM Cortex-M4
- `-mthumb` – The MoaTouch's MCU uses the ARM Thumb-2 instruction set
- `-mfpu=fpv4-sp-d16 -mfloat-abi=softfp` – Use the ARM Cortex-M4's Floating Point Unit (FPU)
- `-fsingle-precision-constant` – treat floating point literals as a `float` type instead of a `double` type as the MoaTouch's FPU only supports single precision floating point.
- `-ffunction-sections -fdata-sections` – Used in collusion with the linker's `-gc-sections` to remove unused code and reduce the executable's size.
- `-std=c++11` – Enabled C++11 features
- `-I"C:\Program Files (x86)\ComfileTools\MoaTouch Tools\include"` – The include folder containing the MoaTouch library header files.

This will create two object files, `HelloWorld.o` and `main.o` to be used by the linker in the next step.

Linking the Object Files

Once the object files (*.o) files are created, we can link them with the MoaTouch library and the toolchain's libraries to produce an *.elf executable The linker is invoked with the C++ compiler in order to automatically link in the toolchain's built-in libraries.

```
arm-none-eabi-g++ -mthumb -mcpu=cortex-m4 -mfpu=fpv4-sp-d16 -mfloat-abi=softfp -Wl,-gc-
sections -L"C:\Program Files (x86)\ComfileTools\MoaTouch Tools\lib" -Wl,--whole-archive
-lMoaTouch -Wl,--no-whole-archive -Wl,-T"C:\Program Files (x86)\ComfileTools\MoaTouch
Tools\linker script\MoaTouch.ld" HelloWorld.o main.o -o HelloWorld.elf
```

- `arm-none-eabi-g++` - Invoke the linker throught the GNU C++ compiler
- `-mcpu=cortex-m4` – The MoaTouch's MCU is an ARM Cortex-M4
- `-mthumb` – The MoaTouch's MCU uses the ARM Thumb-2 instruction set
- `-mfpu=fpv4-sp-d16 -mfloat-abi=softfp` – Use the ARM Cortex-M4's Floating Point Unit (FPU)
- -Wl,--whole-archive -lMoaTouch -Wl,--no-whole-archive
- `-Wl,-gc-sections` – Remove any unused code to reduce the executable's size
- `-L"C:\Program Files (x86)\ComfileTools\MoaTouch Tools\lib" -Wl,--whole-archive -lMoaTouch -Wl,--no-whole-archive` – Link with the MoaTouch static library
- `-Wl,-T"C:\Program Files (x86)\ComfileTools\MoaTouch Tools\linker script\MoaTouch.ld"` – Use the MoaTouch linker script. This linker script tells the linker the structure of the MoaTouch's memory so it can create a compatible executable.
- `HelloWorld.o main.o` – Add the object files created in the compilation stage.

This will create a `HelloWorld.elf` file which is the program's executable.

Create a *.hex File for Uploading

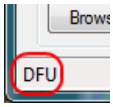
Once `HelloWorld.elf` is created, we need to create a *.hex file that the MoaTouch Control Panel can upload to the MoaTouch.

```
arm-none-eabi-objcopy HelloWorld.elf --target=ihex HelloWorld.hex
```

Upload to the MoaTouch

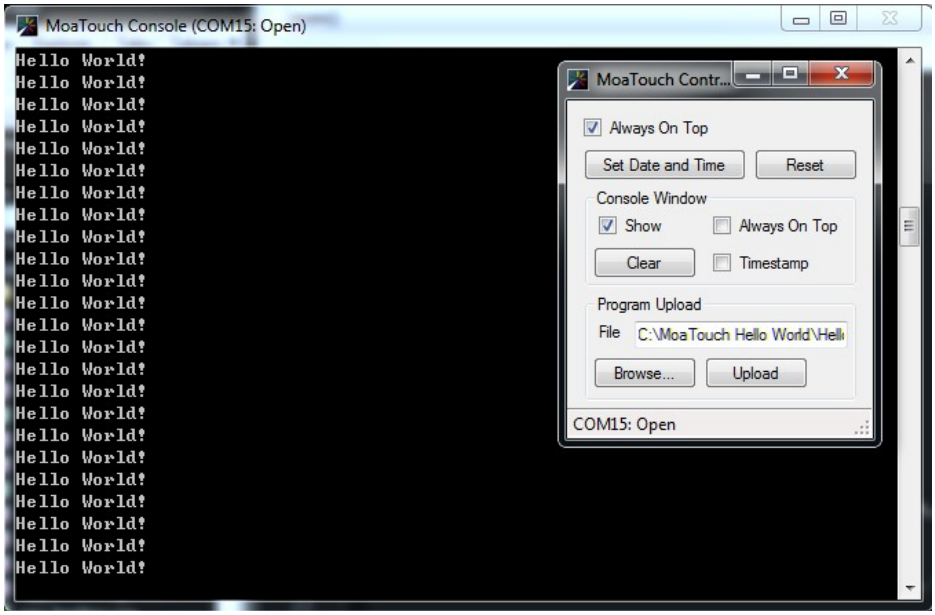
Once `HelloWorld.hex` is created, it can be uploaded to the MoaTouch.

1. Connect the MoaTouch to the host PC via USB and ensure dipswitch 2 is in the ON position to put it in DFU mode.
2. Power on the MoaTouch.
3. Open the MoaTouch Control Panel, and verify that the MoaTouch is connected in DFU mode.



4. Browse to the location of the *.hex file and press the "Upload" button to begin uploading the program to the MoaTouch. The following temporary window will display to show the status of the upload process.

5. When uploading is finished, the program will begin executing immediately. Check the "Show" checkbox in the MoaTouch Control Panel to show the Console Window, and the "Hello World!" message will be seen.

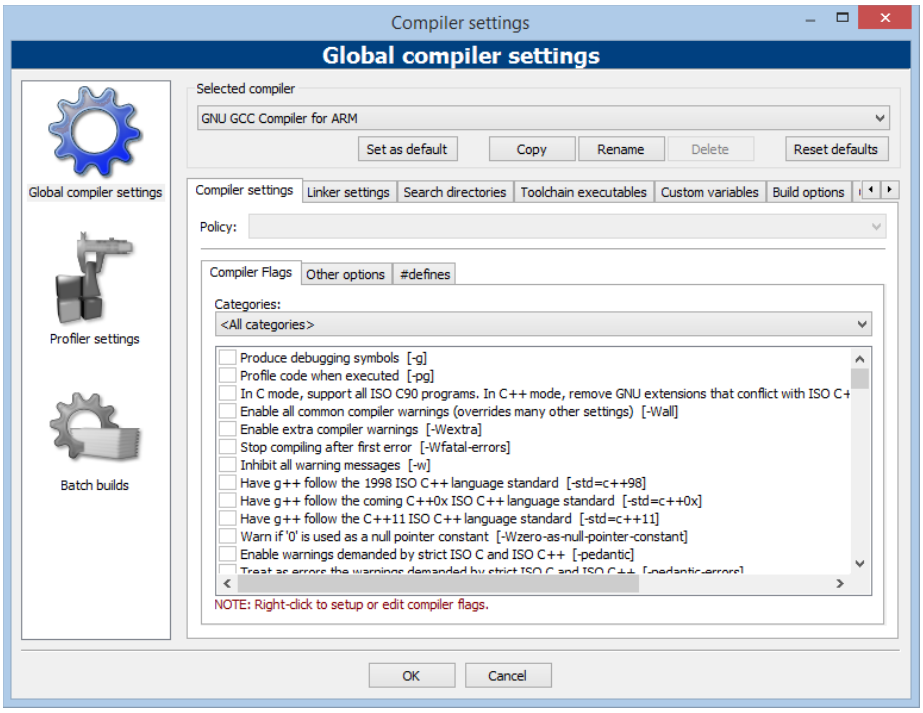


Configuring a Software Integrated Development Environment (IDE)

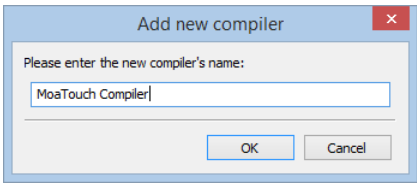
With the information in the previous section one can configure one of the many C/C++ IDEs available. To illustrate this procedure, we will show how to configure Code::Blocks – a free and open source IDE available for download from http://www.codeblocks.org/. Although this procedure is specific to Code::Blocks, readers should be able to adapt this procedure to just about any development environment.

Configuring the Compiler

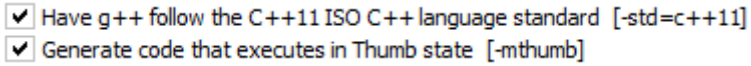
Code::Blocks does not know about the MoaTouch, so we must teach it. From the Code::Blocks menu choose **Settings-->Compiler...** to open the **Global Compiler Settings** window.



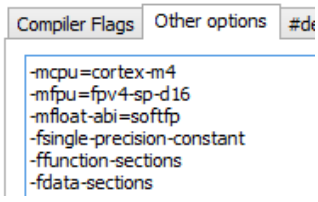
The MoaTouch compile is a variant of the **GNU GCC Compiler for ARM**, so select that compiler and click the **Copy** button. This will prompt you to enter a new name for the compiler. Enter **MoaTouch Compiler.**



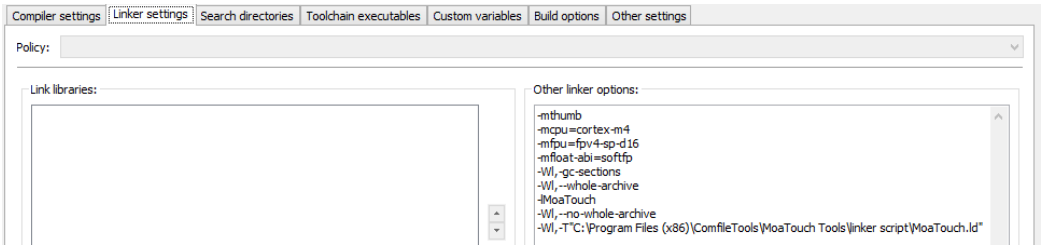
In the list of **Compiler Flags**, check the following options...



... and add the following options to the **Other Flags** tab.

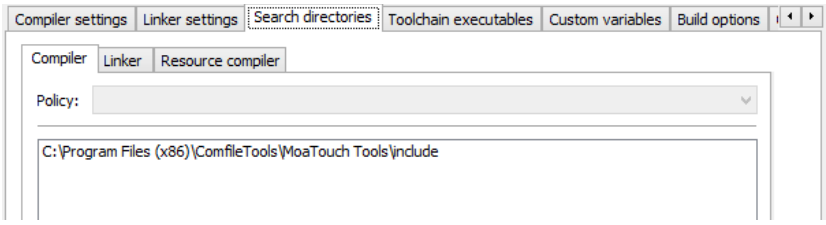


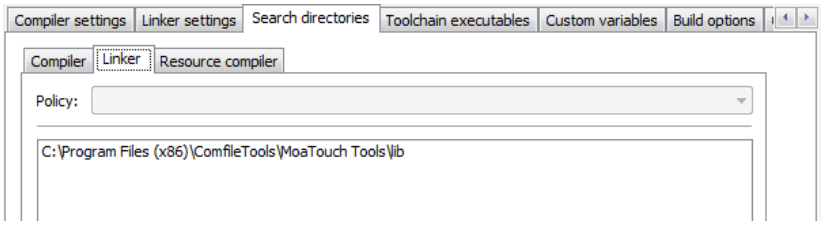
Go to the **Linker Settings** tab and enter the following options.



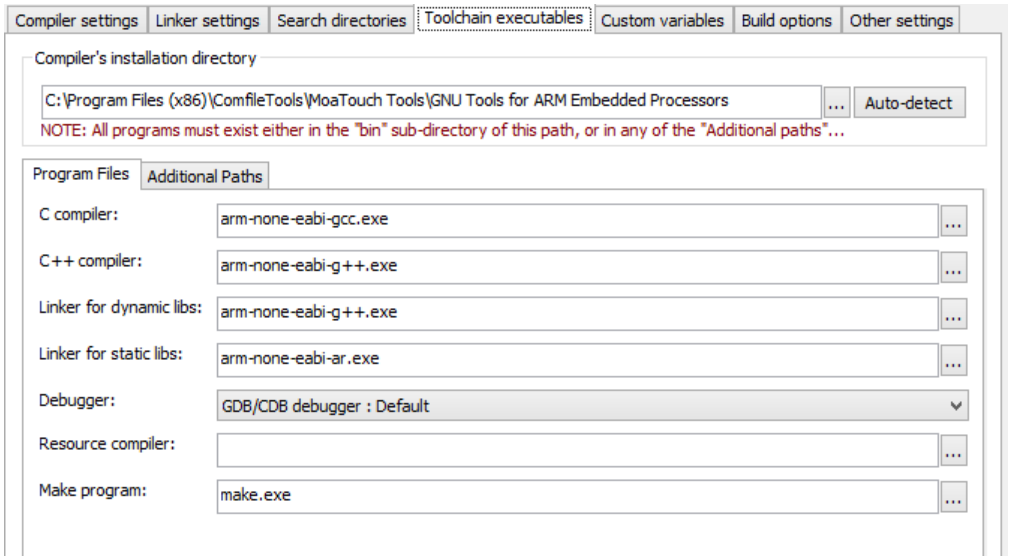
The last option should be the path to the linker script in the MoaTouch Tools installation folder.

Then, go to the **Search directories** tab and enter the MoaTouch Tools include folder and library folder.

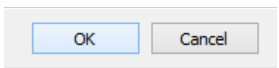




Finally, go t the **Toolchain executables** tab and enter the following.



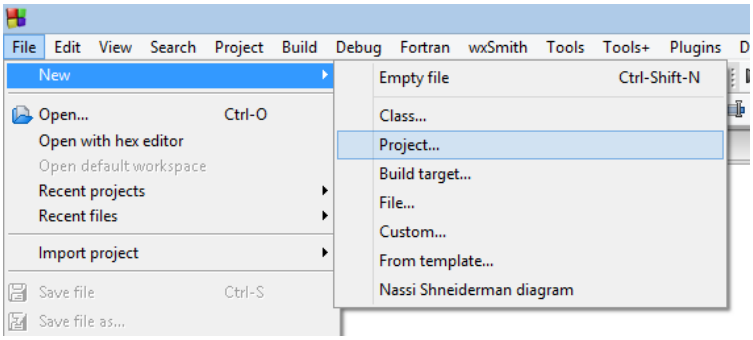
The compiler is now configured. Click the **OK** button to finish.



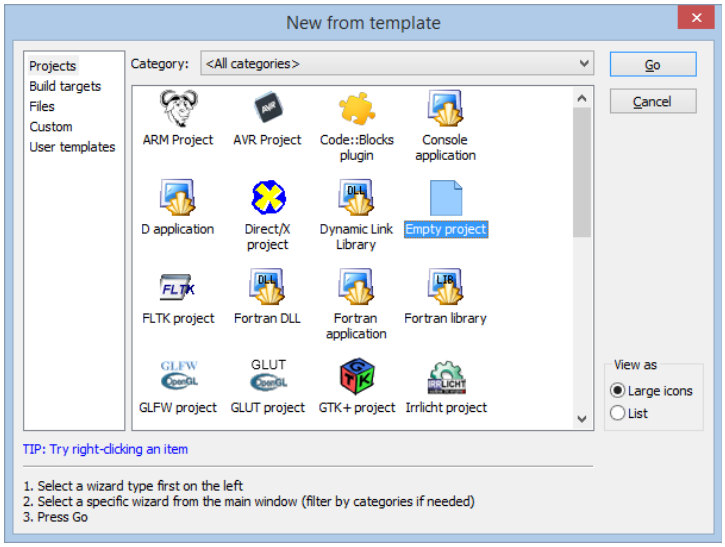
Creating a New Project

This procedure will show how to make a simple "Hello World!" project for the MoaTouch in Code::Blocks

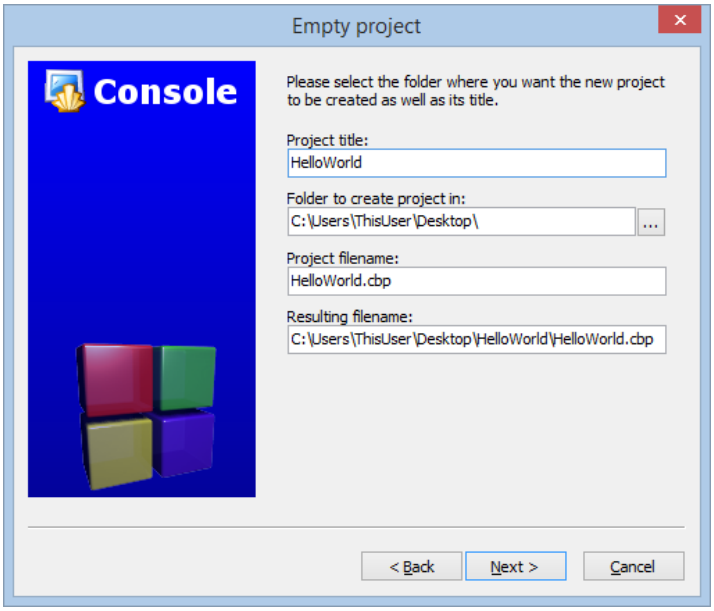
From the Code::Blocks menu select **File-->New-->Project...**



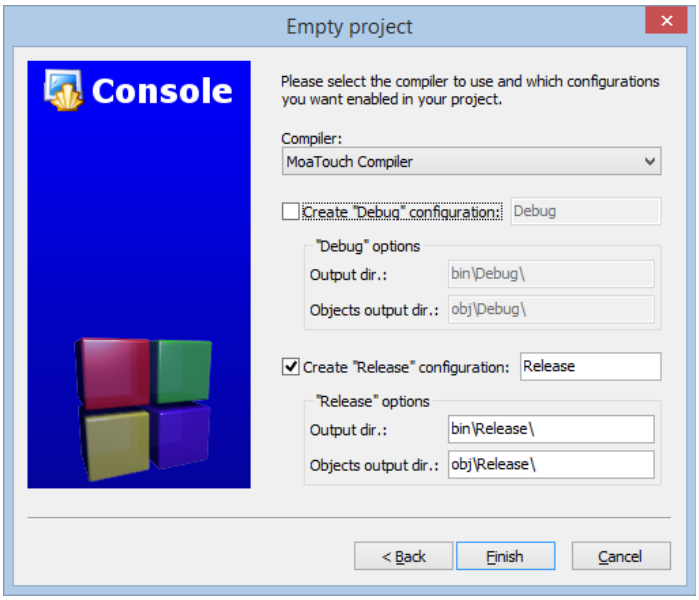
… and select **Empty project** and click the **Go** button.



Enter the project information, and click the **Next** button.

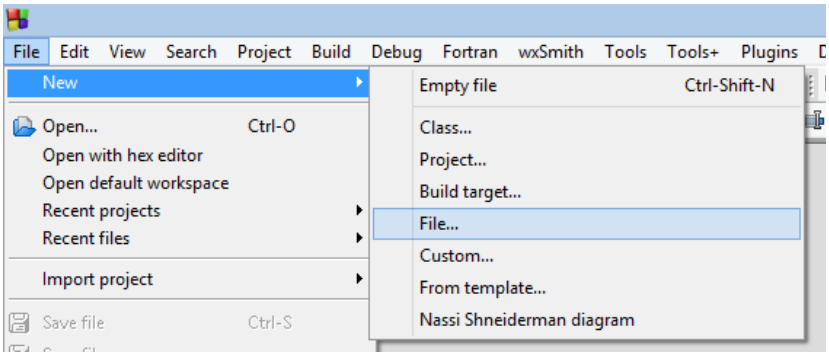


Select the **MoaTouch Compiler** that was configured in the previous section, and create a **Release** configuration. Then click the **Finish** button.

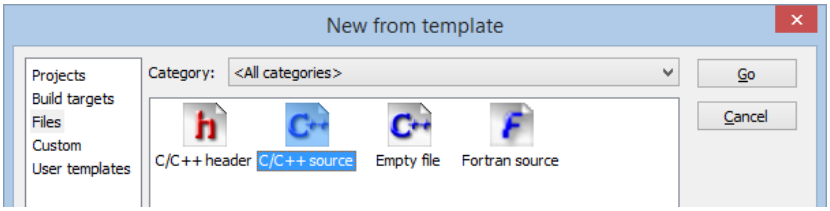


Adding Source Code Files to the Project

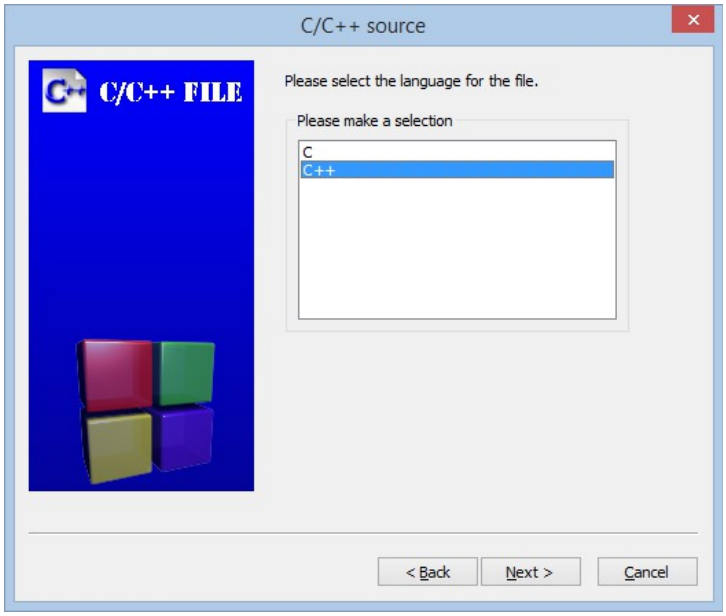
Now we need to add code files to the project. From the Code::Blocks menu, select **File-->New-->File...**



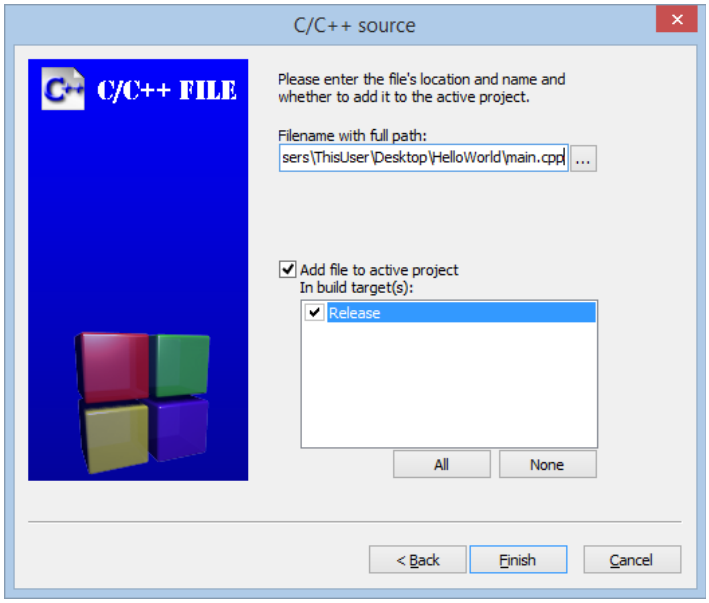
Select **C/C++ source** and click the **Go** button.



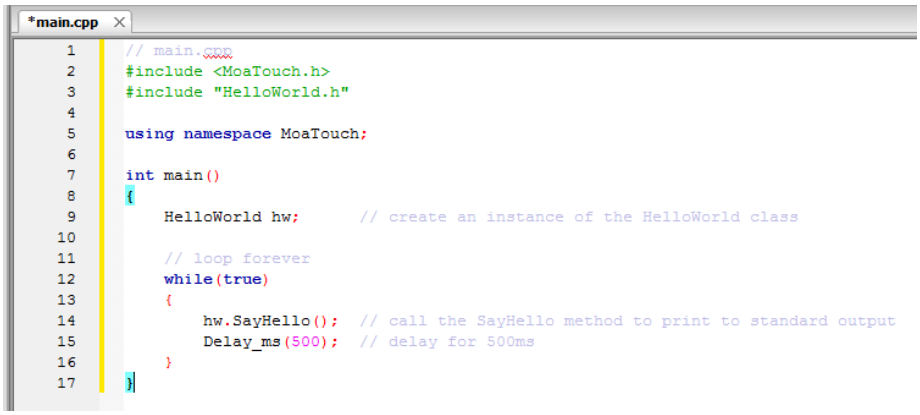
Then select **C++**, and click the **Next** button.



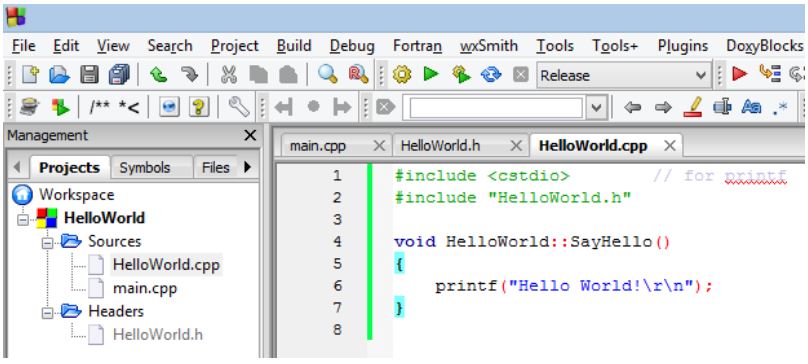
Enter the path to the file (`main.cpp`) and click the **Finish** button.



Edit `main.cpp` as in the previous section.

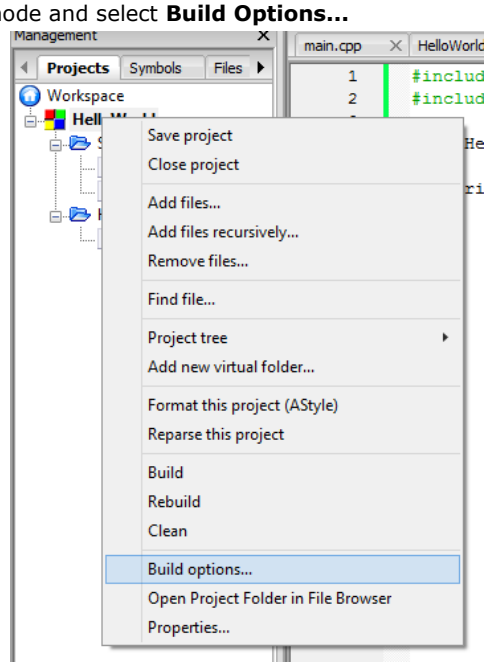


Repeat for `HelloWorld.h` and `HelloWorld.cpp`.

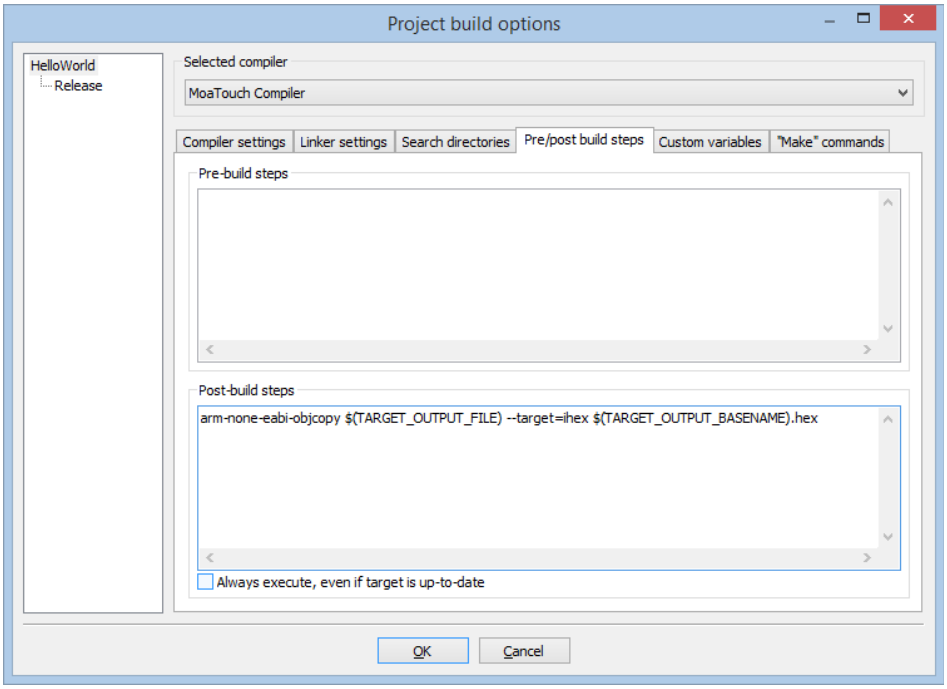


Building the Project

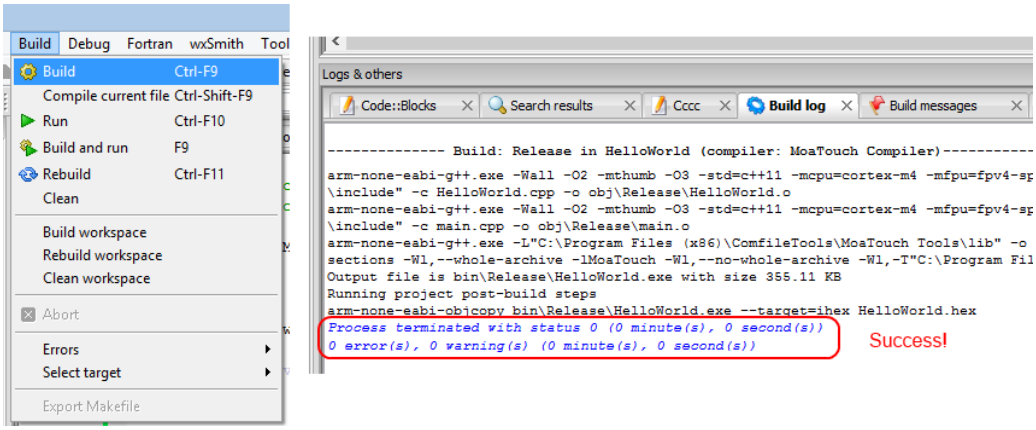
Right-click the **HelloWorld** project node and select **Build Options...**



In the next window, select the HelloWorld node and select the Pre/post build steps tab. Then enter the command to generate the *.hex file as shown below. `$(TARGET_OUTPUT_FILE)` and `$(TARGET_OUTPUT_BASENAME)` are special Code::Blocks variables that refer to the executable file and the executable file name without the extension respectively.



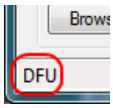
Select **Build-->Build** from the Code::Blocks menu to build the project.



Uploading the Binary to the MoaTouch

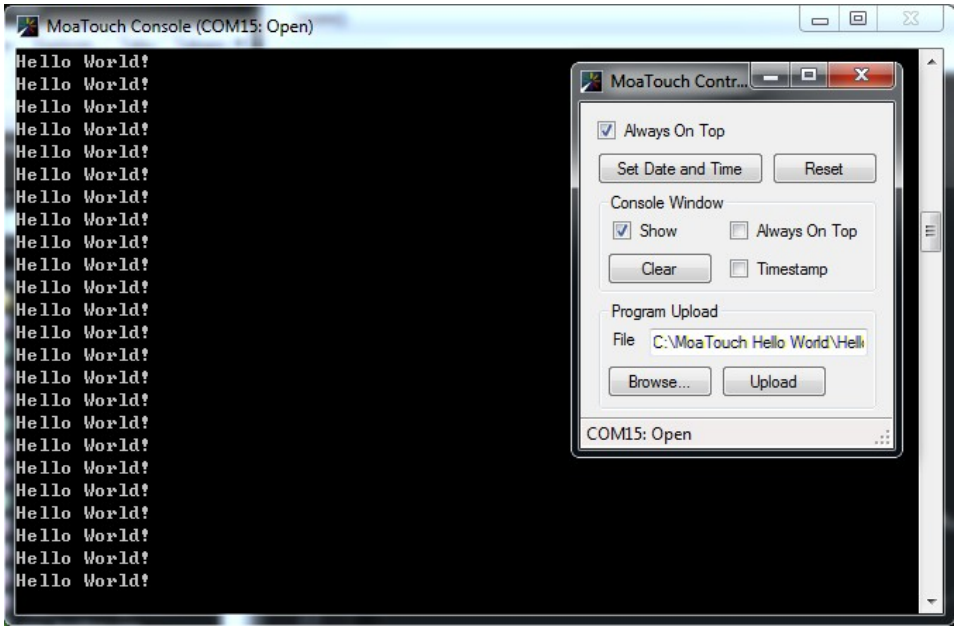
Once `HelloWorld.hex` is created, it can be uploaded to the MoaTouch.

1. Connect the MoaTouch to the host PC via USB and ensure dipswitch 2 is in the ON position to put it in DFU mode.
2. Power on the MoaTouch.
3. Open the MoaTouch Control Panel, and verify that the MoaTouch is connected in DFU mode.



4. Browse to the location of the *.hex file and press the "Upload" button to begin uploading the program to the MoaTouch. The following temporary window will display to show the status of the upload process.

5. When uploading is finished, the program will begin executing immediately. Check the "Show" checkbox in the MoaTouch Control Panel to show the Console Window, and the "Hello World!" message will be seen.



Coordinate System

The MoaTouch features an 800 x 480 pixel display. The X-axis increases from left to right while the Y-axis increases from top to bottom. The origin (0, 0) is the top-left corner of the display.

Coordinates are in units of pixels and are always relative to the active layer's origin. So, if drawn to the background layer, the code…

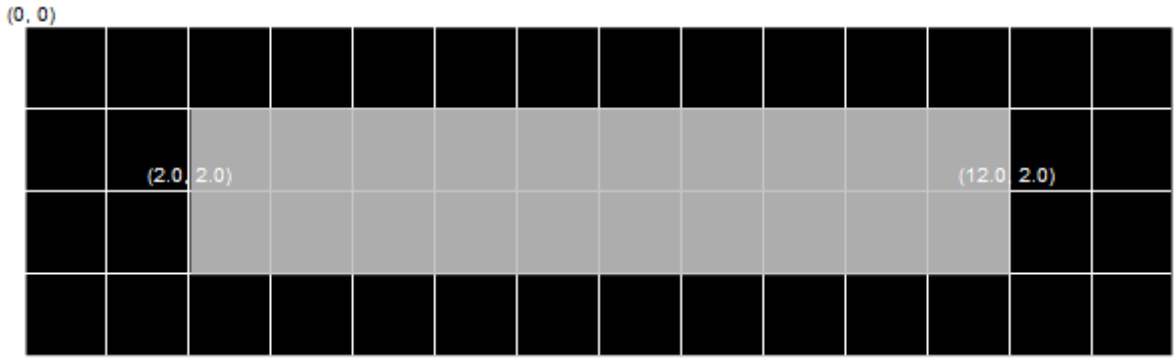
```
SolidBrush white({0xFF, 0xFF, 0xFF});
Stroke stroke(1.0, LineCap::Butt, LineJoin::Round);
lcd.DrawLine(0.0f, 0.0f, 800.0f, 480.0f, stroke, white);
```

…will draw a diagonal line from the top-left corner of the screen to the bottom-right corner of the screen. However, notice that functions for drawing vector graphics employ floating point arguments. This means it's possible to specify coordinates of sub-pixels (e.g. half of a pixel). So, that might beg the question "How does half of a pixel appear?".

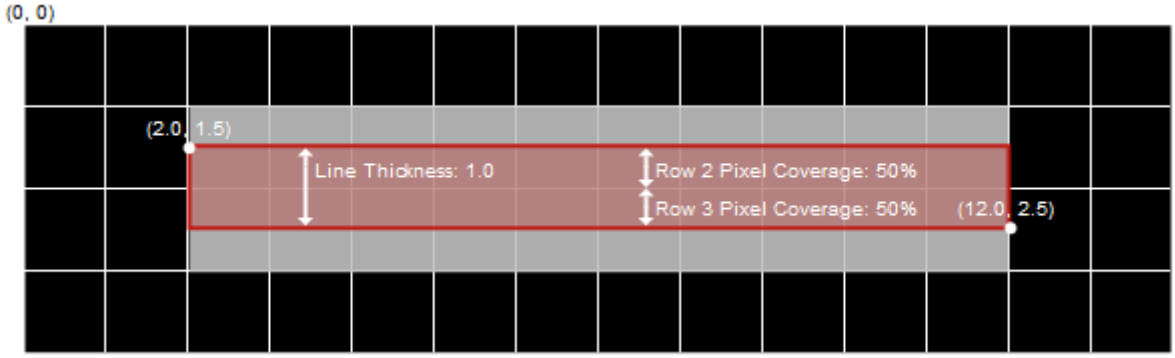
To answer that question, consider the code…

```
SolidBrush white({0xFF, 0xFF, 0xFF});
Stroke stroke(1.0, LineCap::Butt, LineJoin::Round);
lcd.DrawLine(2.0f, 2.0f, 12.0f, 2.0f, stroke, white);
```

One might expect this to draw a bright white horizontal line, 10.0-pixels long, and exactly 1.0-pixel thick. However the line actually appears blurred. Zoomed in, one can see the line is actually 2.0-pixels thick and light-gray in color as shown below.



To understand why this is, one must understand that pixels are not dimensionless; rather they have a width and height. The most common displays have a pixel array of 72 DPI (dots per inch) meaning each pixel is approximately 0.013 inches square (pixels aren't actually square, but, it helps this explanation to think of them as square).

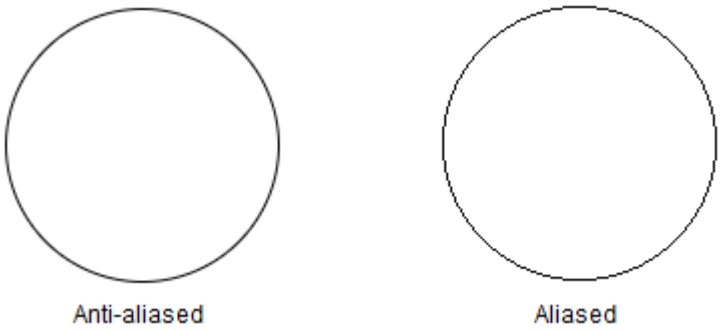


To draw a 1.0-pixel thick horizontal line from (2.0, 2.0) to (12.0, 2.0), that really means drawing a 10.0-pixel wide, 1.0-pixel tall rectangle centered on the line between (2.0, 2.0) and (12.0, 2.0). Therefore, the rectangle's top-left corner is (2.0, 1.5) and bottom-right corner is (12.0, 2.5). This is illustrated on the graph below. The outline of the actual line is shows in red. (NOTE: The ends of the line are clipped exactly at the x-coordinates due to the "Butt" end cap)

One can see that the line partially covers the pixels in rows 2 and 3. That is why the line appears to be two-pixels wide. The pixels in row 2 and 3 are each only half-covered (i.e. the pixel coverage is 50%). Since a single pixel is the smallest individual unit of color, the graphics engine approximates the 50% pixel coverage by making the pixel half as bright (i.e. 50% transparent). That is why, on a black background, the line appears gray instead of white.

In summary, sub-pixels are approximated by varying the pixel's brightness according to the percentage of pixel's area covered by the shape.

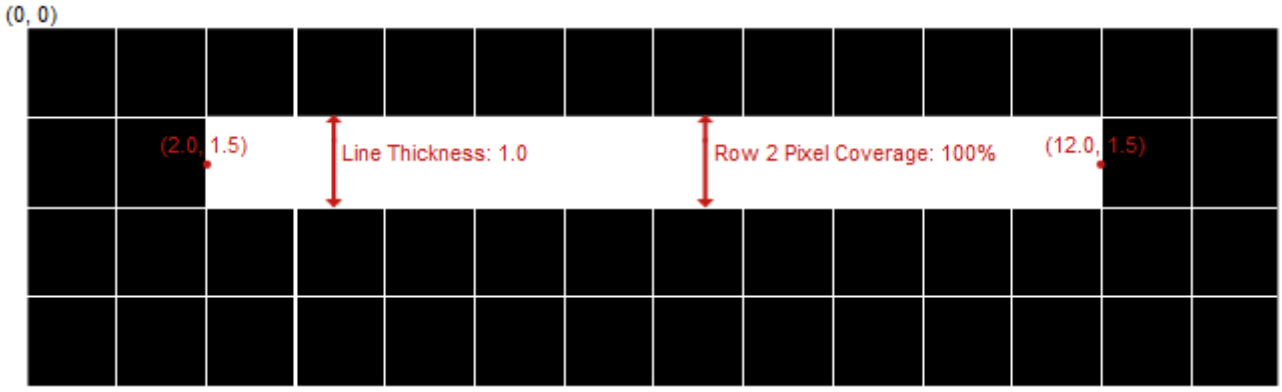
All this is necessary in order to support anti-aliasing, and ensure the visuals appear smooth and professional.



If an application requires a horizontal line that is exactly 1.0-pixel thick and not approximated on the pixel grid, simply shift the line as necessary to achieve 100% pixel coverage.

In the following example, the horizontal line in the previous illustrations is shifted up by 0.5 pixels to get 100% pixel coverage on row 2.

```
SolidBrush white({0xFF, 0xFF, 0xFF});
Stroke stroke(1.0, LineCap::Butt, LineJoin::Round);
lcd.DrawLine(2.0f, 1.5f, 12.0f, 1.5f, stroke, white);
```



Now, because the pixel coverage is 100%, the pixels will be lit at 100% brightness (i.e. 100% opaque), resulting in a bright white horizontal line with a thickness of exactly 1.0 pixel.

Graphics

The MoaTouch can display 3 types of graphics: Vector Graphics, Raster Graphics, and Fonts for text.

Vector Graphics

The majority of the MoaTouch's graphic library repertoire is used to draw vector graphics. Vector graphics consist of geographical primitives (lines, curves, rectangles, circles, etc…) that can be used in isolation or in combination with other visuals to create buttons, graphs, charts, panels, or just about any other visual that can be imagined.

Vector graphics functions can be broken down into 2 categories: stroked visuals and filled visuals. Stroked visuals draw an outline around the shape, and filled visuals have their interior filled with a specific color or pattern.

The following draws two rectangles: one stroked, and one filled. Functions that draw stroked shapes begin with "Draw" and functions that draw filled shapes begin with "Fill". The MoaTouch also contains a `Path` class for drawing arbitrary shapes.

The following example shows two different rectangles: one stroked and one filled.

```
// Clear the background layer
lcd.Clear()

// Draw a stroked rectangle
SolidBrush green({0x00, 0xFF, 0x00});
Stroke stroke(1.0, LineCap::Butt, LineJoin::Miter);
lcd.DrawRectangle(50.0f, 50.0f, 100.0f, 100.0f, stroke, green);

// Draw a filled rectangle
SolidBrush fuchsia({0xFF, 0x00, 0xFF});
lcd.FillRectangle(200.0f, 50.0f, 100.0f, 100.0f, stroke, fuchsia);
```

Vector graphics can be mathematically transformed (rotated, scaled, translated, etc…) without loosing any definition. This makes them particularly useful for rendering dynamic content such as gauge needles and graphs.

Raster Graphics

Raster graphics are simply arrays of plotted pixels. They are mostly used for displaying photographs and other file-based images. The M Display supports two functions for drawing raster graphics: `LCD::DrawImage` and `LCD::SetPixels`. `LCD::DrawImage` can be used to read a PNG file from the SD Card and plot it to the screen at a given location. `LCD::SetPixels` is used to plot one or more pixels at a specified location and can be used to plot images on the screen without needing to read from the SD Card.

The following example shows an image of a gauge being displayed from a PNG file on the SD Card.

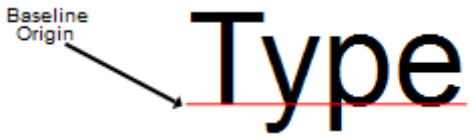
```
// Clear the background layer
lcd.Clear();

// Read the image from the SD Card
File* file = sdCard.Open("gauge.png");

// Display the image
lcd.DrawImage(229.0f, 115.0f, file);

// Clean up
sdCard.Close(file)
```

Fonts and Text

The MoaTouch supports the TrueType font format. TrueType fonts are actually a specialized class of vector graphics, and can, therefore, be scaled, stroked, and filled just like vector graphics.

The shape of the characters (a.k.a. glyphs) are stored in a TrueType font file. The MoaTouch can read these glyphs from a TrueType font file, and draw the characters at a specified location on the screen.

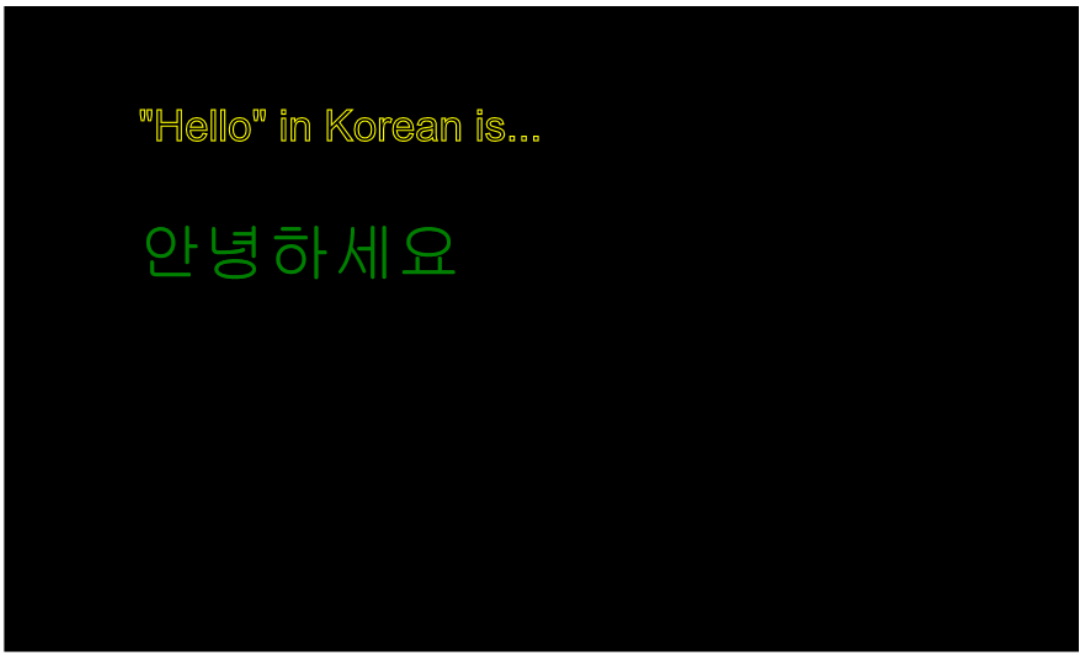
Text is positioned relative to its baseline origin as illustrated below.



The MoaTouch library includes a default embedded font with ASCII characters which can be used without employing any external font files. It is accessed via the `defaultFont` identifier.

This example shows how to stroke and fill text.

```
// Clear the background layer
lcd.Clear();

// Draw yellow outlined text in the default font
SolidBrush yellow({0xFF, 0xFF, 0x00});
lcd.DrawText({100.0f, 100.0f}, defaultFont, "\"Hello\" in Korean is...", 32.0f, yellow);

// Draw green filled text in the gulim font
File* file = sdCard.Open("gulim.ttf");
SolidBrush green({0x00, 0xFF, 0x00});
lcd.FillText({100.0f, 200.0f}, file, "안녕하세요", 48.0f, green);
sdCard.Close(file);
```



When loading a font from the SD Card, every time text is displayed, the glyphs have to be read from the file. This may

be fine for infrequent text updates, but for highly dynamic text, it may be too slow. The MoaTouch supports 2 methods to remedy this.

Method 1: Use the BinaryToC utility to convert the font to a C source file that can be embedded into your program. Please visit the MoaTouch's support website to obtain this utility and its accompanying documentation. The BinaryToC program will convert the font file into a byte array which can be stored permanently stored, along with your program, in the MoaTouch's flash memory.

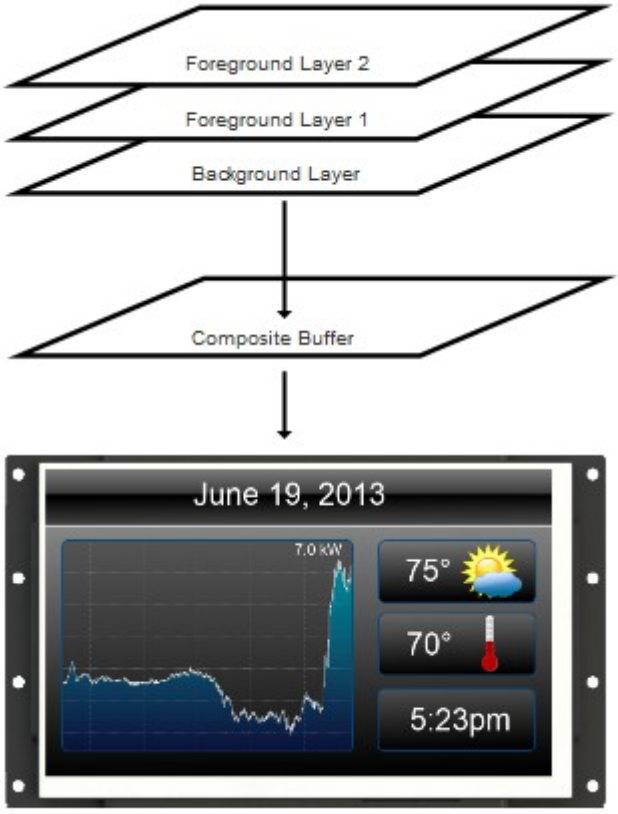
Method 2: Load the font into memory at runtime as shown below:

```
// Open the font file
File* file = sdCard.Open("font.ttf");

// Allocate a memory buffer to hold the font file
uint8_t* buffer = new uint8_t[file->GetSize()];

// Read the entire font file into the memory buffer
file->Read(buffer, file->GetSize());

// Create a memory stream to pass to the font drawing functions
MemoryStream memoryFont(buffer, file->GetSize());

// Close the file as it is no longer needed
sdCard.Close(file);

// Display the text
SolidBrush white({0xFF, 0xFF, 0xFF});
lcd.FillText(100.0f, 100.0f, memoryFont, "Hello", 32.0f, white);

// Deallocate the memory
delete[] buffer;
```

To ensure a font fits in the MoaTouch's limited memory, however, the size of the font file must be reduced to an absolute minimum. To help users reduce the size of font files, Comfile Technology has created a SubsetFont utility that can be used to remove unneeded glyphs from font files. Please see the MoaTouch's support website to obtain this utility, and it's documentation.

Layers

Layers are independent visuals superimposed on one another to create a composite visual. They are useful for separating a visual into logical parts that can be updated independently without impacting other visuals that they may overlap.



At power on, the M Display creates a background layer as the default drawing surface. Additional foreground layers can be created with the `LCD::CreateForegroundLayer` function.

The user can draw, independently, to the background layer or any foreground layer. Using the painter's algorithm (background layer rendered first, foreground layer-1 rendered second…foreground layer-*n* rendered last) the layers are combined into a single frame buffer, the composite buffer. Then on every vertical refresh of the LCD, any changes to the composite buffer are flushed to the screen.

The flushing of the composite buffer to the LCD can be controlled with the `LCD::DisableFlush` and `LCD::EnableFlush` functions. This can be useful to prevent the M Display from displaying part of a composite visual until the entire visual has been completely drawn. It can also improve performance when drawing many visuals, as the M Display doesn't have to waste resources updating the LCD multiple times.

Using Layers to Update Text

Using layers is particularly useful when updating text. Consider the desire to update text on green background.

Attempt 1

If one calls `LCD::FillText` function and attempts to update the text with a second call to `LCD::FillText`, the the second text will be superimposed on the first text as shown below.

```
// Display a green rectangle
SolidBrush green({0x00, 0xFF, 0x00});
lcd.FillRectangle(0.0f, 0.0f, 200.0f, 75.0f, green);

// White Text
File* file = sdCard.Open(&file, "dejavu.ttf");
SolidBrush white({0xFF, 0xFF, 0xFF});

//Display 123
lcd.DrawText(50.0f, 50.0f}, file, "123", 32.0f, white);

//Display 456
lcd.DrawText(50.0f, 50.0f}, file, "456", 32.0f, white);

// close font file
sdCard.Close(file);
```

Attempt 2

If a call to `LCD::Clear` is used in between the two `LCD::FillText` function calls, the background will be erased and a black box will appear, which is also not desirable.

```
// Display a green rectangle
SolidBrush green({0x00, 0xFF, 0x00});
lcd.FillRectangle(0.0f, 0.0f, 200.0f, 75.0f, green);

// White Text
File* file = sdCard.Open(&file, "dejavu.ttf");
SolidBrush white({0xFF, 0xFF, 0xFF});

//Display 123
lcd.DrawText(50.0f, 50.0f}, file, "123", 32.0f, white);

// Clear the box containing the text (black on background layer)
lcd.Clear(40.0f, 20.0f, 70.0f, 40.0f);

// Display 456
lcd.DrawText(50.0f, 50.0f}, file, "456", 32.0f, white);

// close font file
sdCard.Close(file);
```

Attempt 3

However, by separating the green background and the text onto different layers, clearing the foreground layer containing the text will clear the text without impacting the green background layer.

```
// Display a green rectangle
SolidBrush green({0x00, 0xFF, 0x00});
lcd.FillRectangle(0.0f, 0.0f, 200.0f, 75.0f, green);

// White Text
File* file = sdCard.Open(&file, "dejavu.ttf");
SolidBrush white({0xFF, 0xFF, 0xFF});

// Create foreground layer
ForegroundLayer& fg = lcd.CreateForegroundLayer(40.0f, 20.0f, 70.0f, 40.0f);

// White Text
File* file = sdCard.Open(&file, "dejavu.ttf");
SolidBrush white({0xFF, 0xFF, 0xFF});

// Display 123
fg.DrawText(50.0f, 50.0f}, file, "123", 32.0f, white);

// Clear "123" (Transparent for foreground layer)
fg.Clear();

// Display 456
fg.DrawText(50.0f, 50.0f}, file, "456", 32.0f, white);

// close font file
sdCard.Close(file);
```

That's better! One could have simply displayed a new green-filled rectangle to clear the text in this example, but that method falls short for backgrounds that may be images. Images are stored on the SD card and reading from the SD card is orders of magnitude slower than drawing with the CPU. Re-displaying the background image from the SD card for each and every text update, especially if the image were full-screen, would be much too slow to be practical, and this is why the layering feature was created.

Layers and Memory

Creating layers requires memory to hold the state of its pixels. The MoaTouch has 4MB of memory that is shared by all layers, cached fonts, and any other loadable content. When the MoaTouch is first powered on, it creates an 800x480x16-bits = 768KB background layer and an 800x480x16-bits = 768KB composition layer. These layers only need 16-bits per pixel because they are opaque layers. That leaves approximately 2~2.5MB of memory for any foreground layers, cached fonts, and other loadable content.

Foreground layers will reside “on top” of the background layer, and will therefore need an additional byte for the alpha (transparency) component. This byte, in addition to the 16 bits required for the RGB components, results in for a total of 24 bits per pixel. So, as an example, an 800×480 foreground layer will need 800x480x24-bits = 1,152KB of memory.

As can be seen, layers can potentially consume a lot of memory. So, be sure to create layers sparingly, and keep them to as small an area as possible to avoid running out of memory.

Calibrating the Touch Screen

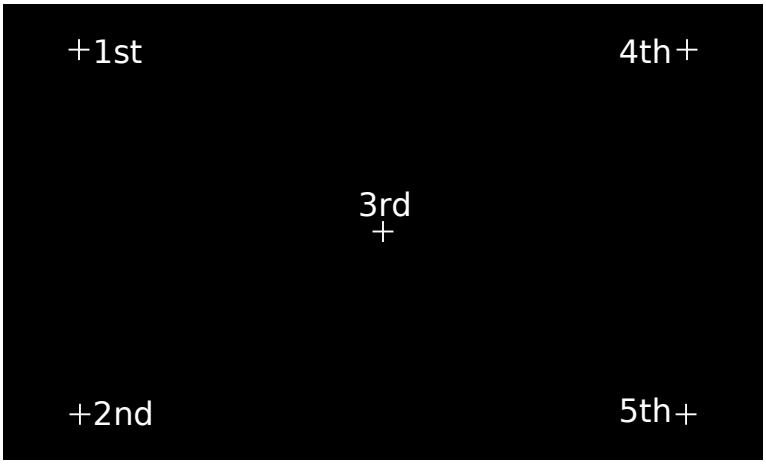
User's may wish to integrate touch screen calibration into their application in a variety of ways, so we have created a special `Touch::Calibrate` method to accommodate this.

The `Touch::Calibrate` function takes a single parameter: a function to draw the point on the screen where the user should touch. The `Touch::Calibration` function will call this function 5 times to complete a 5 point calibration.

Between each point, the MoaTouch will wait for the user to touch the screen. The MoaTouch will take an average of the values it reads while the user is touching the screen. When the user releases their touch, it will remember the average value and display the next point to be touched. After all 5 points have been read the `Touch::Calibration` function will calibrate the touch screen and return. The result of the calibration is saved in a special EEPROM so users will not need to recalibrate between power cycles. If the program changes the screen orientation, the touch screen will need to be recalibrated.

The following is a simple example illustrating the procedure in code:

```
#include <MoaTouch.h>
using namespace MoaTouch;

static void DrawCrosshairs(Point p)
{
    lcd.Clear();   // clear the screen between each point

    // Configure the appears of the crosshairs
    SolidBrush white({0xFF, 0xFF, 0xFF});
    Stroke stroke(2.0, LineCap::Butt, LineJoin::Round);
    float radius = 10.0f;

    // draw the vertical line
    Path line1 = Path::Line({p.x, p.y - radius}, {p.x, p.y + radius});
    lcd.DrawPath(line1, stroke, white);

    // draw the horizontal line
    Path line2 = Path::Line({p.x - radius, p.y}, {p.x + radius, p.y});
    lcd.DrawPath(line2, stroke, white);
}

int main()
{
    touch.Calibrate(DrawCrosshairs);   // perform the calibration
    lcd.Clear();                       // clear the screen

    while(true);                       // loop forever

    return 0;
}
```



Class Reference

`AnalogInput` Class

The MoaTouch has four 12-bit analog inputs. This class provides access to those inputs. Instances of this class can be retreived via the `analogInputs[int index]` or `analogInputs[AnalogInput::Pin pin]` indexers.

`AnalogInput::Pin` Enum

An enumeration for identifying an individual analog input channel. `AnalogInput::A0~A3` corresponds to channels 0~3 respectively.

Methods

`AnalogInput::Pin GetPin() const;`

Gets the AnalogInput::Pin for this instance.

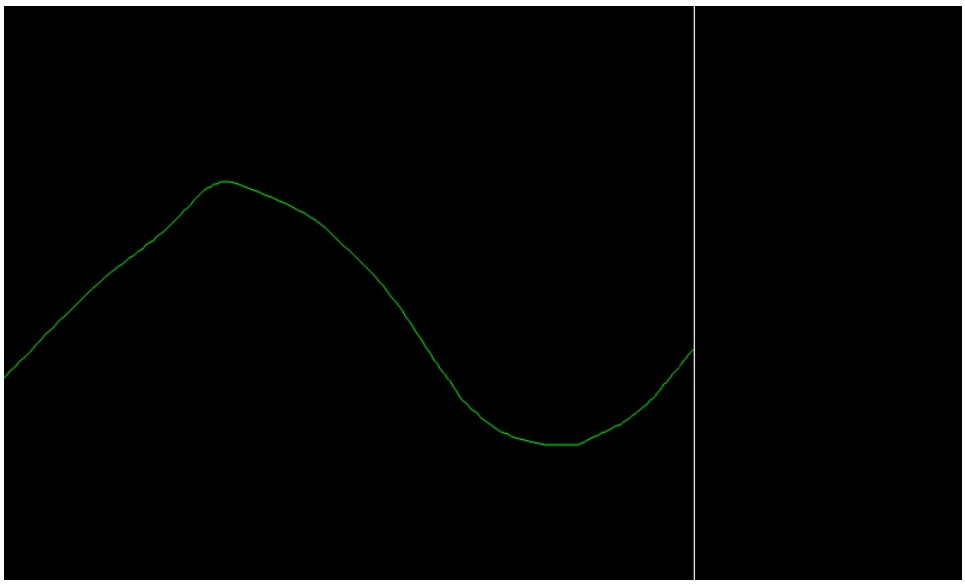
`uint16_t GetValue();`

Perform an analog to digital conversion and return the resulting value.

Example

This example reads from analog input 0 and plots its value as a graph.

```
#include <MoaTouch.h>
using namespace MoaTouch;

int main()
{
    lcd.SetBacklightBrightness(255);            // start off with a blank screen
    lcd.EnableFlush();
    lcd.Clear();
    while(lcd.HasChanges());

    SolidBrush white({0xFF, 0xFF, 0xFF});       // cursor in white
    SolidBrush green({0x00, 0xFF, 0x00});       // value in green
    Stroke line1(1.0f);                         // use a stroke width of 1 pixel

    float cursor = 0.5;                         // the current x coordinate
    float previousValue = 479.0f;               // previous and current y coordinates
    float currentValue = 479.0f;
    while(true)
    {
        // advance the cursor, and wrap around when advancing past the edge of the screen
        cursor = cursor >= 799.5f ? 0.5f : cursor + 1.0f;
        previousValue = currentValue;

        // scale ADC value to the height of the screen
        currentValue = (analogInputs[0].GetValue() * lcd.GetArea().GetHeight()) / 4096.0f;
        currentValue = 480.0f - currentValue;  // because y-axis is top to bottom

        // Clear area of the screen for plotting new point
        lcd.Clear({{cursor, 0}, 2, 480});

        // Draw line between previous point and current point
        Point previousPoint(cursor - 1.0f, previousValue);
        Point currentPoint(cursor, currentValue);
        lcd.DrawLine(previousPoint, currentPoint, line1, green);

        // Draw cursor
        lcd.DrawLine({cursor + 1.0f, 0.0f}, {cursor + 1.0f, 479.0f}, line1, white);

        //Wait for all changes to be drawn
        while(lcd.HasChanges());
    }
    return 0;
}
```



`AnalogInputs` Class

This class simply provides indexers for retrieving the AnalogInput Class instances.

Operators

```
AnalogInput& operator[](AnalogInput::Pin pin) const;
```

Returns the AnalogInput Class instance assigned to the given pin.

```
AnalogInput& operator[](int index) const;
```

Returns the AnalogInput Class instance at the given index. Will throw a `std::out_of_range` exception of `index` is invalid. See the AnalogInput Class Example for a demonstration of this operator.

`Area` Class

A 2-dimensional rectangular area.

Constructors

`Area();`
Creates a area at 0,0 with a width and height of 0

`Area(float x0, float y0, float x1, float y1);`
Creates a rectangular area from four coordinates identifying the points at opposite corners on a diagonal.

`Area(Point p0, Point p1);`
Creates a rectangular area from two points at opposite corners on a diagonal.

`Area(Point topLeft, float width, float height);`
Creates a rectangular area with the top-left corner at `topLeft`, with a width of `width` and a height of `height`.

Methods

`float GetBottom() const;`
Gets the bottom-most y-coordinate of this area

`Point GetBottomLeft() const;`
Gets the point that is the bottom-left corner of this area

`Point GetBottomRight() const;`
Gets the point that is the bottom-right corner of this area

`float GetHeight() const;`
Gets the height of this area

`Area GetIntersection(const Area& area) const;`
Returns an area that is the intersection of the given area and this area

`float GetLeft() const;`
Gets the left-most x-coordinate of this area

`float GetRight() const;`
Gets the right-most x-coordinate of this area

`float GetTop() const;`
Gets the top-most y-coordinate of this area

`Point GetTopLeft() const;`
Gets the point that is the top-left corner of this area

`Point GetTopRight() const;`
Gets the point that is the top-right corner of this area

`float GetWidth() const;`
Gets the width of this area

`bool Intersects(const Point& point) const;`
Returns true if the given point intersects this area

`bool Intersects(const Area& area) const;`
Returns true if the given area and this area intersect one another

`bool IsOffBottomBound(float value) const;`
Returns true if the given coordinate lies below this area

`bool IsOffBottomRightBound(const Point& point) const;`
Returns true if the given point is either to the right of this area, below this area, or both.

`bool IsOffLeftBound(float value) const;`
Returns true if the given coordinate lies to the left of this area

```
bool IsOffRightBound(float value) const;
```
Returns true if the given coordinate lies to the right of this area

```
bool IsOffTopBound(float value) const;
```
Returns true if the given coordinate lies above this area

```
bool IsOffTopLeftBound(const Point& point) const;
```
Returns true if the given point is either to the left of this area, above this area, or both.

```
bool IsWithinHorizontalBounds(float value) const;
```
Returns ture if the given coordinate is greater than or equal to the left coordinate of this area, and less than or equal to the right coordinate of this area.

```
bool IsWithinVerticalBounds(float value) const;
```
Returns true if the given coordinate is greater than or equal to the top coordinate of this area, and less than or equal to the bottom coordinate of this area.

```
void SetHeight(float value);
```
Sets the height of this area

```
void SetLeft(float x);
```
Sets the left-most x coordinate of this area. Setting this value does not change the width or height of this area. It effectively moves the area along the x-axis

```
void SetTop(float y);
```
Sets the top-most y coordinate of this area. Setting this value does not change the width or height of this area. It effectively moves the area along the y-axis

```
void SetTopLeft(const Point value);
```
Sets the top-left corner of this area. Setting this value does not change the width or height of this area. It effectively shifts the area to a new location.

```
void SetWidth(float value);
```
Sets the width of this area

`Brush` Class

This is an abstract class that provide a uniform implementation for all brushes. It is implemented by the SolidBrush Class, the LinearGradientBrush Class, and the RadialGradientBrush Class.

Methods

```
virtual BrushType GetType() const = 0;
```

Gets this instance's `BrushType` for distinguishing it from other brushes.

`Buzzer` Class

The MoaTouch contains a small piezo buzzer that can be used for audio feedback. It is implemented as a singleton class whose instance can be accessed via the `buzzer` identifier.

Methods

`float GetFrequency() const;`

Gets the frequency, in hertz, of the sound to emit.

`bool IsOff() const;`

Returns true if the buzzer is currently off, not emitting sound

`bool IsOn() const;`

Returns true if the buzzer is currently on, emitting sound

`void Off(bool off = true);`

Turns the buzzer off. The `off` argument is optional. If `off` is `true`, the buzzer will turn off. If `off` is `false`, the buzzer will turn on.

`void On(bool on = true);`

Turns the buzzer on. The `on` argument is optional. If `on` is `true`, the buzzer will turn on. If `on` is `false`, the buzzer will turn off.

`void SetFrequency(float value);`

Sets the frequency, in hertz, of the sound to emit.

Example

This example will turn the buzzer on when the screen is touched. It will turn off the buzzer with the the touch is released. The frequency emitted is a function of the x-coordinate of the touch, so touching farther to the right will emit a higher frequency sound, and touching farther to the left will emit a lower frequency sound.

```
#include <MoaTouch.h>
using namespace MoaTouch;

static void OnTouch(Point p)
{
    buzzer.On();                // Turn on the buzzer and set the initial frequency
    buzzer.SetFrequency(p.x);
}

static void OnMove(Point p)
{
    buzzer.SetFrequency(p.x);   // Change the frequency of the buzzer when touch moves
}

static void OnRelease(Point p)
{
    buzzer.Off();               // Turn off buzzer when touch is released
}

int main()
{
    // Assign functions to handle the touch events
    touch.SetOnTouch(OnTouch);
    touch.SetOnMove(OnMove);
    touch.SetOnRelease(OnRelease);

    while(true);                // run forever

    return 0;
}
```

`BrushType` Enumeration

This enumeration is used to distinguish between the different brushes (SolidBrush, LinearGradientBrush, and RadialGradientBrush).

Items

`Solid`

The brush is of type SolidBrush.

`LinearGradient`

The brush is of type LinearGradientBrush.

`RadialGradient`

The brush is of type RadialGradientBrush.

`Color` Class

Represents a 24-bit color (8-bit alpha, 5-bit red, 6-bit green, 5-bit blue).

Constructors

```
Color(uint8_t red = 0x00, uint8_t green = 0x00, uint8_t blue = 0x00, uint8_t alpha = 0xFF);
```
Defaults to opaque black.

```
Color(const Color& color);
```
Copy constructor

Methods

```
void AlphaBlend(const Color565& fgColor, uint8_t alpha);
```
Blends the foreground color, `fgColor`, with this color. `alpha` is the opacity of `fgColor`.

```
uint8_t GetAlpha() const
```
Gets the alpha color component.

```
uint8_t GetBlue() const
```
Gets the blue color component.

```
uint8_t GetGreen() const
```
Gets the green color component.

```
uint8_t GetRed() const
```
Gets the red color component.

```
bool IsOpaque() const
```
Returns true if the color is opaque ((alpha == 0xFF).

```
bool IsTransparent() const
```
Returns true if the color is transparent (alpha == 0x00).

```
Color Lighten(const Color565& color,  float factor);
```
Lightens this color by the given `factor`, and returns the result as a new color

```
void SetAlpha(const uint8_t value)
```
Sets the alpha color component.

```
void SetGreen(const uint8_t value)
```
Sets the green color component.

```
void SetBlue(const uint8_t value)
```
Sets the blue color component.

```
void SetRed(const uint8_t value)
```
Sets the red color component.

`Color565` Class

Represents a 16-bit color (5-bit red, 6-bit green, 5-bit blue).

Constructors

```
Color(uint8_t red = 0x00, uint8_t green = 0x00, uint8_t blue = 0x00);
```
Defaults to black.

```
Color(const Color& color);
```
Copy constructor

Methods

```
void AlphaBlend(const Color565& fgColor, uint8_t alpha);
```
Blends the foreground color, `fgColor`, with this color. `alpha` is the opacity of `fgColor`.

```
uint8_t GetBlue() const
```
Gets the blue color component.

```
uint8_t GetGreen() const
```
Gets the green color component.

```
uint8_t GetRed() const
```
Gets the red color component.

```
Color Lighten(const Color565& color,  float factor);
```
Lightens this color by the given `factor`, and returns the result as a new color

```
void SetGreen(const uint8_t value)
```
Sets the green color component.

```
void SetBlue(const uint8_t value)
```
Sets the blue color component.

```
void SetRed(const uint8_t value)
```
Sets the red color component.

`Console` Class

A class used to print messages to the MoaTouch Control Panel's console. `printf` forwards to the `Console::Print` method. It is implemented as a singleton class whose instance can be accessed via the `console` identifier.

Methods

`void Print(const char* format, ...);`

Prints formatted data. `format` uses the same format specifiers as `printf`.

`void Print(const std::string& s);`

Prints a string. `format` uses the same format specifiers as `printf`.

`void PrintLine(const char* format = nullptr, ...);`

Prints formatted data appended with a new line. `format` uses the same format specifiers as `printf`.

`void PrintLine(const std::string& s);`

Prints string appended with a new line. `format` uses the same format specifiers as `printf`.

`std::string GetNewLine() const;`

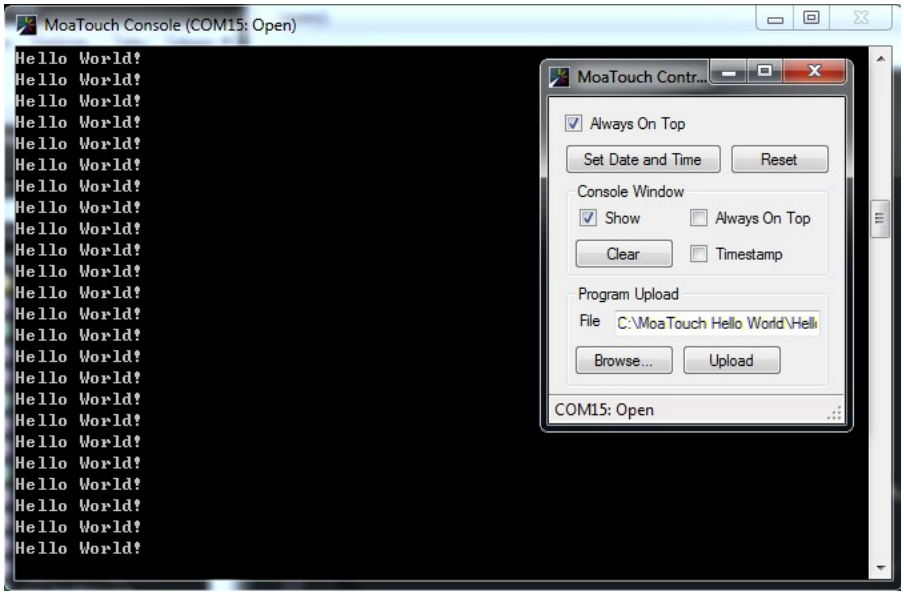
Gets a string to use for new lines.

`void SetNewLine(const string& value = u8"\r\n");`

Sets the string to use for new lines.

Example

```
#include <MoaTouch.h>
using namespace MoaTouch;

int main()
{
    while(true)
    {
        console.PrintLine("Hello World");
        Delay_ms(500);
    }

    return 0;
}
```



`DataBits` Enumeration

This enumeration is used with the SerialPort Class to specify the number of data bits (i.e. word length).

Items

`Bits8 = 8`

8-bit word length

`Bits9 = 9`

9-bit word length

Example

Please see the SerialPort Class Example for a demonstration of this enumeration.

DateTime Class

This class representing a calendar point in time. It is the type returned from the real-time clock (RTC Class).

Constructors

```
DateTime();
```
Instantiates this instance defaulting to January 1, 1900.

```
DateTime(uint16_t  year,  uint8_t month,   uint8_t dayOfMonth,
         uint8_t hours,   uint8_t minutes, uint8_t seconds);
```
Instantiates this instance with the values given. If any value is invalid, `range_error` exception is thrown.

```
DateTime(const DateTime&);
```
Copy constructor

Methods

```
uint8_t GetDayOfMonth() const;
```
Gets the day of the month component (1~31).

```
uint8_t GetHours() const;
```
Gets the hours component (0~23).

```
uint8_t GetMinutes() const;
```
Gets the minutes component (0~59).

```
uint8_t GetMonth() const;
```
Gets the month component (1~12).

```
uint8_t GetSeconds() const;
```
Gets the seconds component (0~59).

```
uint16_t GetYear() const;
```
Gets the year component (1900~2099).

```
void SetDayOfMonth(uint8_t value);
```
Sets the day of the month component (1~31). `range_error` exception is thrown if value is invalid.

```
void SetHours(uint8_t value);
```
Gets the hours component (0~23). `range_error` exception is thrown if value is invalid.

```
void SetMinutes(uint8_t value);
```
Sets the minutes component (0~59). `range_error` exception is thrown if value is invalid.

```
void SetMonth(uint8_t value);
```
Sets the month component (1~12). `range_error` exception is thrown if value is invalid.

```
void SetSeconds(uint8_t value);
```
Sets the seconds component (0~59). `range_error` exception is thrown if value is invalid.

```
void SetYear(uint16_t value);
```
Sets the year component (1900~2099). `range_error` exception is thrown if value is invalid.

Operators

bool operator==(const DateTime& rhs);
Returns true if this value is equal to `rhs`

bool operator!=(const DateTime& rhs);
Returns true if this value is not equal to `rhs`

Example

```
#include <MoaTouch.h>
#include <string>

using namespace std;
using namespace MoaTouch;

//Add leading zeros to numbers less than 10
static string PadZeros(uint8_t value)
{
    return value < 10 ? "0" + ToString(value) : ToString(value);
}

int main()
{
    lcd.SetBacklightBrightness(255);             // start off with a blank screen
    lcd.EnableFlush();
    lcd.Clear();
    while(lcd.HasChanges());

    // keep track of previous value so we only update when the value changes
    DateTime previousValue;
    while(true)
    {
        // Get the current date and time
        DateTime currentValue = rtc.Get();

        // Don't update screen unless time has changed
        if (currentValue != previousValue)
        {
            // Disable updating changes on the screen.  Helps avoid flickering
            lcd.DisableFlush();

            // Clear the screen
            lcd.Clear();

            // remember this value for comparisons next time
            previousValue = currentValue;

            // Convert value to a string representation
            string s = ToString(currentValue.GetYear())
                + "-" + ToString(currentValue.GetMonth())
                + "-" + ToString(currentValue.GetDayOfMonth())
                + " " + ToString(currentValue.GetHours())
                + ":" + PadZeros(currentValue.GetMinutes())
                + ":" + PadZeros(currentValue.GetSeconds());

            SolidBrush white({0xFF, 0xFF, 0xFF});
            lcd.FillText({100.0f, 100.0f}, defaultFont, s, 32.0f, white);

            // Enable updating changes on the screen
            lcd.EnableFlush();
        }

        // Wait for all changes to be drawn
        while(lcd.HasChanges());
    }
    return 0;
}
```

Delay Functions

The following functions provide the ability to pause execution for a short period of time. They simply forward to the corresponding static functions in the Stopwatch Class and therefore carry the same limitation, that they cannot be used to delay for durations longer than 50 seconds.

```
void Delay_us(float us);
```

Delay for the given number of microseconds

```
void Delay_ms(float ms);
```

Delay for the given number of milliseconds

```
void Delay_s(float s);
```

Delay for the given number of seconds

`DigitalInput` Class

The MoaTouch has 4 GPIO digital inputs. This class provides access to those inputs. Instances of this class can be retrieved via the `digitalInputs[int index]` or `digitalInputs[DigitalInput::Pin pin]` indexers.

`DigitalInput::Pin` Enum

An enumeration for identifying an individual digital input channel (`I0` ~ `I3`)

Methods

```
Pin GetPin() const;
```
Returns the `DigitalInput::Pin` assignment for this instance.

```
bool IsHigh() const ;
```
Returns true if the input is a logic high.

```
bool IsLow() const;
```
Returns true if the input is a logic low.

```
bool IsOff() const;
```
Returns true if the input is a logic low.

```
bool IsOn() const;
```
Returns true if the input is a logic high

```
bool IsInterruptEnabled() const;
```
Returns true if the interrupt for this instance is enabled. The interrupt is disabled by default.

```
void EnableInterrupt(const bool enable = true);
```
Enables the interrupt for this instance. The interrupt is disabled by default.

```
void DisableInterrupt(const bool disable = true);
```
Disables the interrupt for this instance. The interrupt is disabled by default.

```
Edge GetInterruptEdge() const;
```
Gets the edge trigger for this instance's interrupt

```
void SetInterruptEdge(const Edge edge);
```
Sets the edge trigger for this instance's interrupt

```
std::function<void()> GetOnInterrupt() const;
```
Gets the function to be called when this instance's interrupt is triggered

```
void SetOnInterrupt(std::function<void()> value);
```
Sets the function to be called when this instance's interrupt is triggered.

`DigitalInputs` Class

This class simply provides indexers for retrieving the DigitalInputs Class instances.

Operators

```
DigitalInput& operator[](DigitalInput::Pin pin) const;
```

Returns the DigitalInputs Class instance assigned to the given pin.

```
DigitalInput& operator[](int index) const;
```

Returns the DigitalInputs Class instance at the given index. Will throw a `std::out_of_range` exception of `index` is invalid.

`DigitalOutput` Class

The MoaTouch has 4 GPIO digital outputs. This class provides access to those outputs. Instances of this class can be retrieved via the `digitalOutputs[int index]` or `digitalOutputs[DigitalOutput::Pin pin]` indexers.

`DigitalOutput::Pin` Enum

An enumeration for identifying an individual digital input channel (`O0` ~ `O3`)

Methods

```
static DigitalOutput& GetInstance(Pin pin);
```
Get the instance of this class for the given pin

```
Pin GetPin() const;
```
Returns the `DigitalOutput::Pin` assignment for this instance

```
bool IsHigh() const ;
```
Returns true if the input is a logic high.

```
bool IsLow() const;
```
Returns true if the input is a logic low.

```
bool IsOn() const;
```
Returns true if the current output state is logic high.

```
bool IsOff() const;
```
Returns true if the current output state is logic low.

```
void On(bool on = true);
```
Sets the current output state to logic high (on == true) or logic low (on == false)

```
void Off(bool off = true);
```
Sets the current output state to logic low (off == true) or logic high (off == false)

```
void High(bool high = true);
```
Sets the current output state to logic high (high == true) or logic low (high == false)

```
void Low(bool low = true);
```
Sets the current output state to logic low (low == true) or logic high (low == false)

```
void Toggle();
```
Toggles the current output state

`DigitalOutputs` Class

This class simply provides indexers for retrieving the DigitalOutput Class instances.

Methods

```
uint32_t GetState() const;
```

Returns the state of all 32 outputs. The least significant bit is bit `ExDigitalOutput::Pin::O0`.

```
void SetState(uint32_t value);
```

Sets the state of all 32 outputs. The least significant bit is bit `ExDigitalOutput::Pin::O0`.

Operators

```
DigitalOutput& operator[](DigitalOutput::Pin pin) const;
```

Returns the DigitalOutput Class instance assigned to the given pin.

```
DigitalOutput& operator[](int index) const;
```

Returns the DigitalOutput Class instance at the given index. Will throw a `std::out_of_range` exception of `index` is invalid.

`Ethernet` Class

This class is used to configure the MoaTouch's Ethernet port. This class is implemented as a singleton and can be accesses using the `ethernet` identifier.

Use the Socket Class to perform TCP/IP communication. The MoaTouch has 4 sockets.

The MoaTouch does not have a DHCP client, but there is no reason one couldn't be built with the MoaTouch's features.

The MoaTouch does not support UDP.

Methods

```
void GetHardwareAddress(uint8_t* value) const;
```
Gets the hardware address of the MoaTouch as an array of 6 bytes

```
void SetHardwareAddress(uint8_t* value);
```
Sets the hardware address of the MoaTouch as an array of 6 bytes

```
void SetHardwareAddress(uint8_t oct5, uint8_t oct4, uint8_t oct3, uint8_t oct2, uint8_t oct1,
uint8_t oct0);
```
Sets the hardware address of the MoaTouch using individual octets

```
uint32_t GetIPAddress() const;
```
Gets the IP address of the MoaTouch as a 32-bit unsigned integer

```
void SetIPAddress(uint32_t value);
```
Sets the IP address of the MoaTouch as a 32-bit unsigned integer

```
void SetIPAddress(uint8_t oct3, uint8_t oct2, uint8_t oct1, uint8_t oct0);
```
Sets the IP address of the MoaTouch using individual octets

```
uint32_t GetSubnetMask() const;
```
Gets the subnet mask of the MoaTouch as a 32-bit unsigned integer

```
void SetSubnetMask(uint32_t value);
```
Sets the subnet mask of the MoaTouch as a 32-bit unsigned integer

```
void SetSubnetMask(uint8_t oct3, uint8_t oct2, uint8_t oct1, uint8_t oct0);
```
Sets the subnet mask of the MoaTouch using individual octets

```
uint32_t GetGateway() const;
```
Gets the gateway address as a 32-bit unsigned integer

```
void SetGateway(uint32_t value);
```
Sets the gateway address as a 32-bit unsigned integer

```
void SetGateway(uint8_t oct3, uint8_t oct2, uint8_t oct1, uint8_t oct0);
```
Sets the gateway address using individual octets

```
Socket& GetSocket(Socket::Number number);
```
Get the instance of the given socket. See the Socket Class.

Example

See the Socket Class Example for a demonstration of this class.

`ExDigitalInput` Class

The MoaTouch has 32 expansion digital inputs. This class provides access to those inputs. Instances of this class can be retrieved via the `exDigitalInputs[int index]` or `exDigitalInputs[ExDigitalInput::Pin pin]` indexers.

`ExDigitalInput::Pin` Enum

An enumeration for identifying an individual digital input channel (`I0` ~ `I3`)

Methods

```
Pin GetPin() const;
```

Returns the `DigitalInput::Pin` assignment for this instance.

```
bool IsHigh() const ;
```

Returns true if the input is a logic high.

```
bool IsLow() const;
```

Returns true if the input is a logic low.

```
bool IsOff() const;
```

Returns true if the input is a logic low.

```
bool IsOn() const;
```

Returns true if the input is a logic high

`ExDigitalInputs` Class

This class simply provides indexers for retrieving the ExDigitalInput Class instances.

Methods

```
std::function<void(uint32_t)> GetOnChanged() const;
```

Gets the function to be called when one of the inputs' state changes.

```
void SetOnChanged(std::function<void(uint32_t)> value);
```

Sets the function to be called when one of the inputs' state changes.

```
uint32_t GetState() const;
```

Returns the state of all 32 inputs. The least significant bit is bit `ExDigitalInput::Pin::I0`.

Operators

```
ExDigitalInput& operator[](ExDigitalInput::Pin pin) const;
```

Returns the ExDigitalInput Class instance assigned to the given pin

```
ExDigitalInput& operator[](int index) const;
```

Returns the ExDigitalInput Class instance at the given index. Will throw a `std::out_of_range` exception of `index` is invalid.

`ExDigitalOutput` Class

The MoaTouch has 32 expansion digital outputs. This class provides access to those outputs. Instances of this class can be retrieved via the `exDigitalOutputs[int index]` or `exDigitalOutputs[ExDigitalOutput::Pin pin]` indexers.

`ExDigitalOutput::Pin` Enum

An enumeration for identifying an individual digital input channel (`O0` ~ `O31`)

Methods

```
static ExDigitalOutput& GetInstance(Pin pin);
```
Get the instance of this class for the given pin

```
Pin GetPin() const;
```
Returns the `ExDigitalOutput::Pin` assignment for this instance

```
bool IsHigh() const ;
```
Returns true if the input is a logic high.

```
bool IsLow() const;
```
Returns true if the input is a logic low.

```
bool IsOn() const;
```
Returns true if the current output state is logic high.

```
bool IsOff() const;
```
Returns true if the current output state is logic low.

```
void On(bool on = true);
```
Sets the current output state to logic high (on == true) or logic low (on == false)

```
void Off(bool off = true);
```
Sets the current output state to logic low (off == true) or logic high (off == false)

```
void High(bool high = true);
```
Sets the current output state to logic high (high == true) or logic low (high == false)

```
void Low(bool low = true);
```
Sets the current output state to logic low (low == true) or logic high (low == false)

```
void Toggle();
```
Toggles the current output state

`ExDigitalOutputs` Class

This class simply provides indexers for retrieving the ExDigitalOutput Class instances.

Operators

```
ExDigitalOutput& operator[](ExDigitalOutput::Pin pin) const;
```

Returns the ExDigitalOutput Class instance assigned to the given pin.

```
ExDigitalOutput& operator[](int index) const;
```

Returns the ExDigitalOutput Class instance at the given index. Will throw a `std::out_of_range` exception of `index` is invalid.

`File` Class

This class is used with the SDCard Class to do file I/O. It inherits from the FiniteStreamReader Class.

Methods

`void Flush();`
Flush any cached data to disk. Throws a `std::runtime_error` on failure.

`size_t GetCurrentPosition();`
Reports the current position in the file. Throws a `std::runtime_error` on failure.

`size_t GetSize() const;`
Gets the size of the file in bytes

`size_t Read(void *const buffer, const size_t numOfBytes);`
Read `numOfBytes` from the current position in the file and store the data read in `buffer`. Throws a `std::runtime_error` on failure. Be aware that the MoaTouch's stack memory does not support DMA, so `buffer` must be allocated on the heap with the `new` operator or `malloc`. This method will also advance the current position in the file by `numOfBytes`.

`int ReadByte();`
Reads a single byte from the file. Returns a negative number on failure.

`void Seek(const size_t position);`
Moves the file cursor to the given position. Throws a `std::runtime_error` on failure.

`void Truncate();`
Truncates the file, removing its contents, but not the file itself. Throws a `std::runtime_error` on failure.

`size_t Write(const void* buffer, const size_t numOfBytes);`
Reads `numOfBytes` from `buffer` and writes the data to the current position in the file. Throws a `std::runtime_error` on failure. Be aware that the MoaTouch's stack memory does not support DMA, so `buffer` must be allocated on the heap with the `new` operator or `malloc`.

`size_t WriteByte(uint8_t value);`
Writes a single byte, `value`, to the file. Returns the number of bytes actually written.

Example

This example will copy the contents of one file into another.

```
#include <MoaTouch.h>

using namespace MoaTouch;

int main()
{
    while(!sdCard.IsInserted());                    // wait card to be inserted

    sdCard.Mount();                                 // Mount the filesystem

    const char* file1Name = "File1.txt";
    const char* file2Name = "File2.txt";
    if (sdCard.Exists(file1Name))
    {
        if (!sdCard.Exists(file2Name))              // Ensure file 2 exists
        {
            sdCard.Create(file2Name);
        }

        console.PrintLine("Copying...");

        File& file1 = *sdCard.Open(file1Name);      // Open file 1
```

```
        File& file2 = *sdCard.Open(file2Name);     // Open file 2

        file2.Truncate();                          // clear file 2's contents

        uint8_t* buffer = new uint8_t[512];        // temporary buffer to use while copying
                                                   // must use heap as stack doesn't support
                                                   // DMA
        try
        {
            while(!file1.IsEndOfFile())            // copy until the end of the file
            {
                // read data from file1
                size_t bytesRead = file1.Read(buffer, sizeof(buffer));

                // write data to file 2
                file2.Write(buffer, bytesRead);
            }
        }
        catch(const exception& ex)
        {
            console.Print("Error: ");
            console.PrintLine(ex.what());
        }

        delete[] buffer;                           // dealocate temporary buffer

        sdCard.Close(file2);                       // close file 2
        sdCard.Close(file1);                       // close file 1
    }

    sdCard.Unmount();                              // unmount the filesystem

    while(true);                                   // run forever

    return 0;
}
```

Console window output:

```
Copying...
Finished
```

`FileInfo` Class

This class is used with the SDCard Class to get information about an individual file or folder.

Methods

```
std::string GetName() const;
```

Gets the file/folder name

```
size_t GetSize() const;
```

Returns the size of the file in bytes

```
bool IsFile() const;
```

Returns true if this is a file

```
bool IsFolder() const;
```

Return true if this is a folder

Example

Please see the SDCard Class Example for a demonstration using this class.

`FiniteStreamReader` Class

The is an abstract class to provide a uniform interface for accessing streams of data. It is implemented by the File Class and the MemoryStream Class.

Methods

```
virtual size_t GetCurrentPosition() = 0;
```

Reports the current position in the stream.

```
virtual size_t GetSize() const = 0;
```

Reports the total number of bytes in the stream.

```
size_t Read(void *const buffer, const size_t numOfBytes);
```

Read `numOfBytes` from the current position in the stream and store the data read in `buffer`. This method will also advance the current position in the stream by `numOfBytes`.

```
virtual void Seek(const size_t position) = 0;
```

Moves to the given position in the stream.

`ForegroundLayer` Class

This class is used to create independent areas of the screen and combine them using the painter's algorithm. `ForegroundLayer`'s can be created with the `LCD::CreateForegroundLayer` method. It inherits from the Layer Class.

Miscellaneous Methods

This class implements all of the Layer Class's Miscellaneous Methods in addition to the methods below.

```
bool IsEnabled() const;
```
Returns true if this layer is enabled

```
void Enable(bool enable);
```
Enables this layer so its contents are flushed to the LCD screen

```
void Disable(bool disable);
```
Enables this layer so its contents are not to the LCD screen

```
void SetLeft(const PixelCoordinate x);
```
Relocates the left coordinate of this layer, effectively moving the layer horizontally

```
void SetTop(const PixelCoordinate y);
```
Relocates the top coordinate of this layer, effectively moving the layer vertically

```
void SetTopLeft(const PixelPoint& value);
```
Relocates the top-left coordinate of this layer, effectively moving the layer.

```
float GetAbsoluteX(float relativeX) const;
```
Given an x-coordinate relative to this layer's origin, returns the absolute x-coordinate relative to the screen's origin

```
float GetRelativeX(float absoluteX) const;
```
Given an x-coordinate relative to the screen's, returns the x-coordinate relative to this layer's origin

```
float GetAbsoluteY(float relativeY) const;
```
Given a y-coordinate relative to this layer's origin, returns the absolute y-coordinate relative to the screen's origin

```
float GetRelativeY(float absoluteY) const;
```
Given a y-coordinate relative to the screen's, returns the y-coordinate relative to this layer's origin

```
Point GetAbsolutePoint(const Point& relativePoint) const;
```
Given a point relative to this layer's origin, returns the absolute point relative to the screen's origin

```
Point GetRelativePoint(const Point& absolutePoint) const;
```
Given an absolute point relative to the screen's, returns the point relative to this layer's origin

```
Area GetAbsoluteArea(const Area& relativeArea) const;
```
Given an area relative to this layer's origin, returns an absolute area relative to the screen's origin

```
Area GetRelativeArea(const Area& absoluteArea) const;
```
Given an absolute area relative to the screen's, returns an area relative to this layer's origin

Raster Graphics Methods

This class implements all of the Layer Class's Raster Graphics Methods.

Vector Graphics Methods

This class implements all of the Layer Class's Vector Graphics Methods.

Text Methods

This class implements all of the Layer Class's Text Methods.

`GradientStop` Class

This class is by the LinearGradientBrush Class and RadialGradientBrush Class to specify the brush's coordinates and colors.

Constructors

```
GradientStop();
```

Instantiates a new uninitialized instance of this class.

```
GradientStop(const Point& point, const Color& color);
```

Instantiates a new instance of this class at the specified point and with the given color. Coordinates are relative to the active layer.

```
GradientStop(const GradientStop& source);
```

Copy constructor

Methods

```
Point GetPoint() const;
```

Gets the point identifying the location of this instance.

```
void SetPoint(const Point& value);
```

Sets the point identifying the location of this instance.

```
Color GetColor() const;
```

Gets the color for this instance.

```
void SetColor(const Color value);
```

Sets the color for this instance.

Operators

```
GradientStop& operator=(const GradientStop& source);
```

Copy assignment operator

Example

See the LinearGradientBrush Example and the RadialGradientBrush Example for a demonstration using this class.

`Layer` Class

This is an abstract base class to provide uniform drawing features to both background layers and foreground layers. It is implemented by the LCD Class and the ForegroundLayer Class.

Miscellaneous Methods

```
PixelArea GetArea() const;
```

Get the LCD's dimensions

```
PixelPoint GetOrigin() const;
```

Gets the origin (top-left corner) of this layer

```
uint16_t GetWidth() const;
```

Gets the width of this layer

```
uint16_t GetHeight() const;
```

Get the height of this layer

```
void Clear();
```

Clear the background layer

```
void Clear(const PixelArea& area);
```

Clear the given area of the background layer

Raster Graphics Methods

```
void DrawImage(const PixelPoint& topLeft, FiniteStreamReader& image);
```

Plot an image to the screen with positioned with its top-left corner at `topLeft`. Please see Raster Graphics for a demonstration of this method.

```
void SetPixel(const PixelPoint& point, const Color& color);
```

Set the pixel at the given `point` to the given `color`.

```
void SetPixels(const PixelPoint& firstPoint, const std::vector<Color>& colors);
```

Set a consecutive row of pixels starting at point `firstPoint` to the given `colors`.

Vector Graphics Methods

```
void DrawLine(const Point& p0, const Point& p1, const Stroke& stroke, const Brush& brush);
```

Draw a line from `p0` to `p1` using the given `stroke` and `brush`. Please see Example - Lines Curves for a demonstration of this method.

```
void DrawRectangle(const Area& area, const Stroke& stroke, const Brush& brush);
```

Draw the perimeter of a rectangle encompassing the given `area`, using the given `stroke` and `brush`. Please see Example - Rectangles for a demonstration of this method.

```
void DrawRectangle(const Area& area, float cornerRadius, const Stroke& stroke, const Brush& brush);
```

Draw the perimeter of a rounded rectangle encompassing the given `area`, with a uniform `cornerRadius`, using the given `stroke` and `brush`. Please see Example - Rectangles for a demonstration of this method.

```
void DrawRectangle(const Area& area, float topLeftCornerRadius, float topRightCornerRadius, float
bottomLeftCornerRadius, float bottomRightCornerRadius, const Stroke& stroke, const Brush& brush);
```

Draw the perimeter of a rounded rectangle encompassing the given `area`, with nonuniform corner radii, using the given `stroke` and `brush`.

```
void FillRectangle(const Area& area, const Brush& brush);
```

Fill a rectangle encompassing the given `area`, using the given `brush`. Please see Example - Rectangles for a demonstration of this method.

```
void FillRectangle(const Area& area, float cornerRadius, const Brush& brush);
```

Fill a rounded rectangle encompassing the given `area`, with a uniform `cornerRadius`, using the given `brush`. Please see Example - Rectangles for a demonstration of this method.

```
void FillRectangle(const Area& area, float topLeftCornerRadius, float topRightCornerRadius, float
bottomLeftCornerRadius, float bottomRightCornerRadius, const Brush& brush);
```

Fill a rounded rectangle with nonuniform corner radii, using the given `brush`.

```
void DrawEllipse(const Point& center, float horizontalRadius, float verticalRadius, const Stroke&
stroke, const Brush& brush);
```

Draw the perimeter of an ellipse at `center`, with the given `horizontalRadius` and `verticalRadius`, using the given `stroke` and `brush`. See Example - Circles and Ellipses for a demonstration of this method.

```
void FillEllipse(const Point& center, float horizontalRadius, float verticalRadius, const Brush&
brush);
```

Fill an ellipse at `center`, with the given, `horizontalRadius` and `verticalRadius` using the given `brush`. See Example - Circles and Ellipses for a demonstration of this method.

```
void DrawCircle(const Point& center, float radius, const Stroke& stroke, const Brush& brush);
```

Draw the perimeter of a circle at `center`, with the given `radius`, using the given `stroke` and `brush`. See Example - Circles and Ellipses for a demonstration of this method.

```
void FillCircle(const Point& center, float radius, const Brush& brush);
```

Fill a circle at `center` with the given radius using the given `brush`. See Example - Circles and Ellipses for a demonstration of this method.

```
void DrawArc(const Point& center, float radius, float startAngle, float sweep, const Stroke& stroke,
const Brush& brush);
```

Draw an circular arc starting at `startAngle`, traveling `sweep` degrees, using the given `stroke` and `brush`. Positive numbers sweep in the clockwise direction. Negative numbers sweep in the counter-clockwise direction. See Example - Arcs for a demonstration of this method.

```
void DrawArc(const Point& center, float horizontalRadius, float verticalRadius, float startAngle,
float sweep, const Stroke& stroke, const Brush& brush);
```

Draw an elliptical arc starting at `startAngle`, traveling `sweep` degrees, using the given `stroke` and `brush`. Positive numbers sweep in the clockwise direction. Negative numbers sweep in the counter-clockwise direction. See Example - Arcs for a demonstration of this method.

```
void DrawCurve(const Point& from, const Point& control, const Point& to, const Stroke& stroke, const
Brush& brush);
```

Draw a bezier curve `from` point from to point `to` with control point `control` using the given `stroke` and `brush`. Please see Example - Lines Curves for a demonstration of this method.

```
void DrawCurve(const Point& from, const Point& control0, const Point& control1, const Point& to,
const Stroke& stroke, const Brush& brush);
```

Draw a bezier curve from point `from` to point `to` with control points `control0` and `control1` using the given `stroke` and `brush`.

```
void DrawPath(const Path& path, const Stroke& stroke, const Brush& brush);
```

Draw the given `path` using the given `stroke` and `brush`. See the Path Class Pie Section Example and Curve Example and for a demonstration of this method.

```
void FillPath(const Path& path, const Brush& brush);
```

Fill the given `path` using the given `brush`. See the Path Class Pie Section Example and Curve Example and for a demonstration of this method.

```
Area MeasurePath(const Path& path, const Stroke& stroke);
```

Get the encompassing bounding box of the `path` if it were drawn with the given `stroke`.

Text Methods

```
void DrawText(const Point& baselineOrigin, FiniteStreamReader& font, const std::u32string& text,
float size, const Stroke& stroke, const Brush& brush);
void DrawText(const Point& baselineOrigin, FiniteStreamReader& font, const std::u32string& text,
float size, float angle, const Stroke& stroke, const Brush& brush);
void DrawText(const Point& baselineOrigin, FiniteStreamReader& font, const std::string& text, float
size, const Stroke& stroke, const Brush& brush);
void DrawText(const Point& baselineOrigin, FiniteStreamReader& font, const std::string& text, float
size, float angle, const Stroke& stroke, const Brush& brush);
void DrawText(const Point& baselineOrigin, FiniteStreamReader* font, const std::u32string& text,
float size, const Stroke& stroke, const Brush& brush);
void DrawText(const Point& baselineOrigin, FiniteStreamReader* font, const std::u32string& text,
float size, float angle, const Stroke& stroke, const Brush& brush);
void DrawText(const Point& baselineOrigin, FiniteStreamReader* font, const std::string& text, float
size, const Stroke& stroke, const Brush& brush);
void DrawText(const Point& baselineOrigin, FiniteStreamReader* font, const std::string& text, float
size, float angle, const Stroke& stroke, const Brush& brush);
```

Draw the outline of text. Please see Fonts and Text for a demonstration of this method.

- `baselineOrigin` – The point on the baseline to begin drawing the first character.
- `font` – The TrueType font to use to draw the text.
- `text` – The text to display as either a UTF-8 string (`std::string`) or a UTF-32 string (`std::u32string`)
- `size` – The size of the font in pixels.
- `angle` – The angle, in degrees, at which to rotate the text
- `stroke` – The characteristics of the line used to draw the outline of the text
- `brush` – The characteristics of the coloring used to draw the outline of the text

```
void FillText(const Point& baselineOrigin, FiniteStreamReader& font, const std::u32string& text,
float size, const Brush& brush);
void FillText(const Point& baselineOrigin, FiniteStreamReader& font, const std::u32string& text,
float size, float angle, const Brush& brush);
void FillText(const Point& baselineOrigin, FiniteStreamReader& font, const std::string& text, float
size, const Brush& brush);
void FillText(const Point& baselineOrigin, FiniteStreamReader& font, const std::string& text, float
size, float angle, const Brush& brush);
void FillText(const Point& baselineOrigin, FiniteStreamReader* font, const std::u32string& text,
float size, const Brush& brush);
void FillText(const Point& baselineOrigin, FiniteStreamReader* font, const std::u32string& text,
float size, float angle, const Brush& brush);
void FillText(const Point& baselineOrigin, FiniteStreamReader* font, const std::string& text, float
size, const Brush& brush);
void FillText(const Point& baselineOrigin, FiniteStreamReader* font, const std::string& text, float
size, float angle, const Brush& brush);
```

Draws the text, filling the interior of each character with the given brush. Please see Fonts and Text for a demonstration of this method.

- `baselineOrigin` – The point on the baseline to begin drawing the first character.
- `font` – The TrueType font to use to draw the text.
- `text` – The text to display as either a UTF-8 string (`std::string`) or a UTF-32 string (`std::u32string`)
- `size` – The size of the font in pixels.
- `angle` – The angle, in degrees, at which to rotate the text
- `brush` – The characteristics of the coloring used to fill the text

```
Area MeasureText(const Point& baselineOrigin, FiniteStreamReader& font, const std::u32string& text,
float size);
Area MeasureText(const Point& baselineOrigin, FiniteStreamReader& font, const std::u32string& text,
float size, const Stroke& stroke);
Area MeasureText(const Point& baselineOrigin, FiniteStreamReader& font, const std::string& text,
float size);
Area MeasureText(const Point& baselineOrigin, FiniteStreamReader& font, const std::string& text,
float size, const Stroke& stroke);
```

Get the encompassing bounding box of the text if it were drawn to the screen. This can be used to help align and

position text on the screen.

- `baselineOrigin` – The point on the baseline to begin the first character.
- `font` – The TrueType font to use to determine the shape the text.
- `text` – The text to measure as either a UTF-8 string (`std::string`) or a UTF-32 string (`std::u32string`)
- `size` – The size of the font in pixels.
- `stroke` – The characteristics of the line used to outline the shape of the characters.

LCD Class

This class is used display 2-D graphics on the MoaTouch's LCD screen. All drawing functions draw to the main background layer. Additional layers can be created with the `CreateForegroundLayer` method. This class is implemented as a singleton that can be accessed via the `lcd` identifier. This class implements the Layer Class and all of its methods.

Miscellaneous Methods

This class implements all of the Layer Class's Miscellaneous Methods in addition to the methods below.

```
ScreenOrientation GetOrientation() const;
```

Gets the screen orientation. See the ScreenOrientation Enumeration.

```
void SetOrientation(ScreenOrientation orientation);
```

Sets the screen orientation. See the ScreenOrientation Enumeration.

```
void DisableFlush(bool disable = true);
```

Disable flushing the frame buffer to the LCD. If you have many items to draw, use this to prevent displaying each item on the screen until all items have been drawn. Using this command in combination with `EnableFlush` can improve performance and prevent display anomalies while drawing.

```
void EnableFlush(bool enable = true);
```

Enable flushing the frame buffer to the LCD.

```
bool IsFlushEnabled() const;
```

Returns whether or not flushing from the frame buffer to the LCD is enabled.

```
bool HasChanges() const;
```

Returns true if there are changes in the frame buffer that have not yet been flushed to the LCD

```
void SetBacklightBrightness(const uint8_t value);
```

Set the brightness of the LCD backlight (0 = off, 255 = maximum brightness).

```
ForegroundLayer& CreateForegroundLayer(const PixelArea& area);
```

Create a new foreground layer at the specified area. This command dynamically allocates memory to hold the layer's frame buffer, so be aware that creating too may layers or too large of a layer can result in memory errors. See Layers for a demonstration of this method.

```
void DestroyForegroundLayer(ForegroundLayer& layer);
```

Destroy the given foreground layer, deallocating its frame buffer.

```
ForegroundLayer& GetForegroundLayer(unsigned int index) const;
```

Get the foreground layer at the given index.

Raster Graphics Methods

This class implements all of the Layer Class's Raster Graphics Methods.

Vector Graphics Methods

This class implements all of the Layer Class's Vector Graphics Methods.

Text Methods

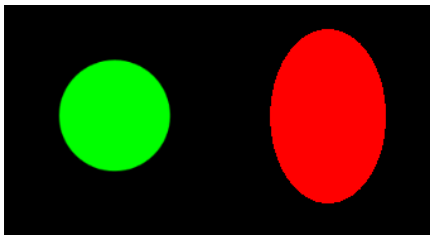
This class implements all of the Layer Class's Text Methods.

Example – Arcs

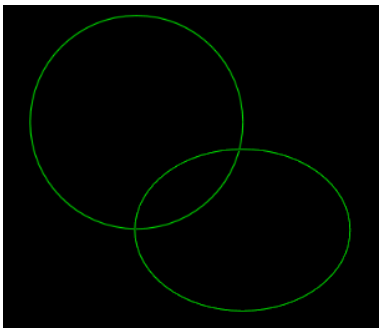
```
#include <MoaTouch.h>

using namespace MoaTouch;

int main()
{
    // draw a green circular arc 1.0 pixel thick
    Stroke stroke1(1.0f);
    SolidBrush green({0x00, 0xFF, 0x00});
    lcd.DrawArc({100.0f, 100.0f}, 50.0f, 45.0f, 235.0f, stroke1, green);

    // draw a fuchsia elliptical arc 3.0 pixels thick
    Stroke stroke3(3.0f);
    SolidBrush fuchsia({0xFF, 0x00, 0xFF});
    lcd.DrawArc({300.0f, 100.0f}, 100.0f, 50.0f, 110.0f, 270.0f, stroke3, fuchsia);

    // run forever
    while(true);

    return 0;
}
```

Example – Circles and Ellipses

```
#include <MoaTouch.h>

using namespace MoaTouch;

int main()
{
    // draw a green circle
    SolidBrush green({0x00, 0xFF, 0x00});
    lcd.FillCircle({100.0f, 100.0f}, 50.0f, green);

    // draw a red ellipse
    SolidBrush red({0xFF, 0x00, 0x00});
    lcd.FillEllipse({300.0f, 100.0f}, 50.0, 75.0f, red);

    // run forever
    while(true);

    return 0;
}
```

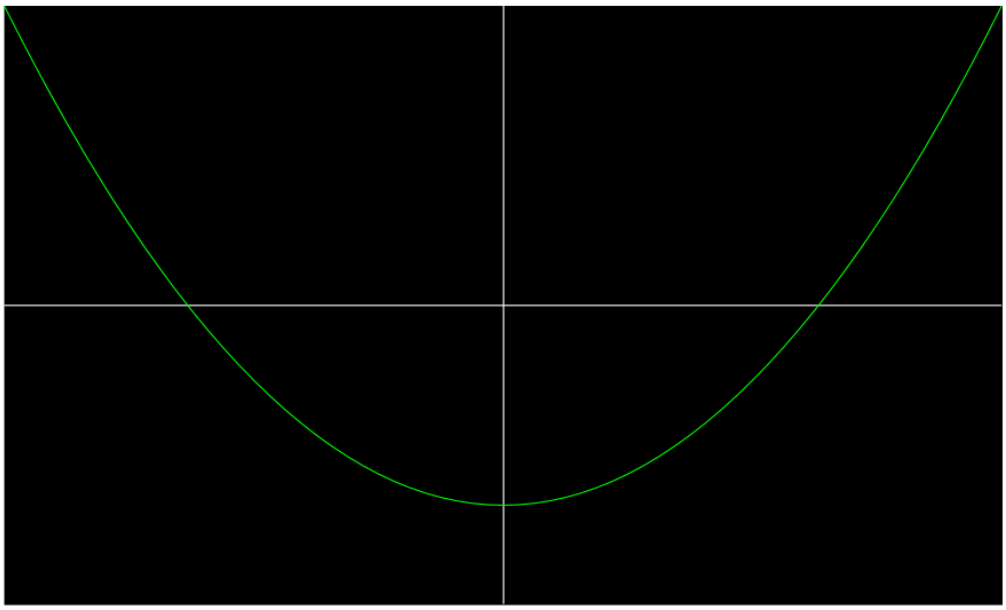


```
#include <MoaTouch.h>

using namespace MoaTouch;

int main()
{
    // Green with a stroke, 1 pixel wide
    SolidBrush green({0x00, 0xFF, 0x00});
    Stroke stroke(1.0f);

    // draw a circle
    lcd.DrawCircle({200.0f, 200.0f}, 100.0f, stroke, green);

    // draw an ellipse
    lcd.DrawEllipse({300.0f, 300.0f}, 100.0, 75.0f, stroke, green);

    // run forever
    while(true);

    return 0;
}
```



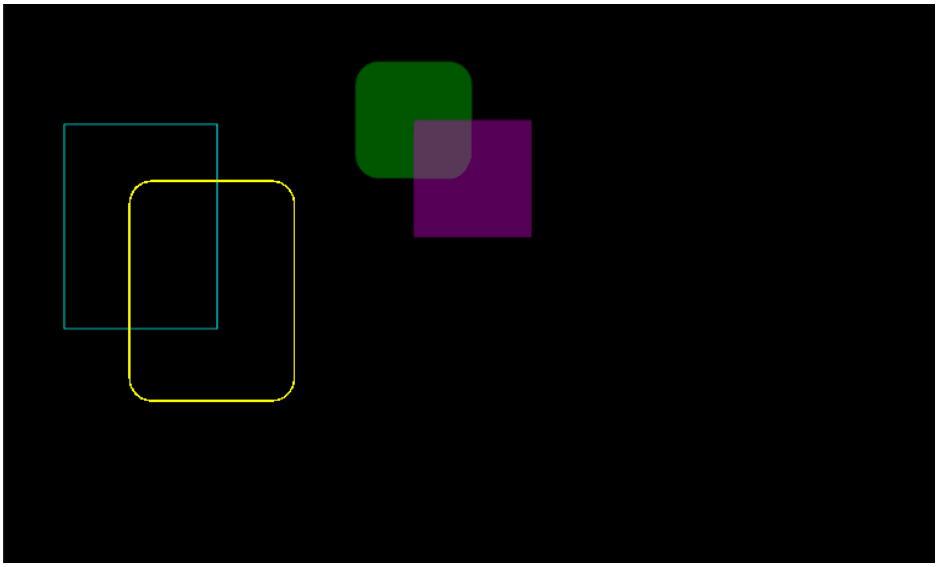
Example – Lines Curves

```
#include <MoaTouch.h>

using namespace MoaTouch;

int main()
{
    // All lines 1px thick
    Stroke stroke(1.0f);

    // Draw white vertical and horizontal lines
    SolidBrush white({0xFF, 0xFF, 0xFF});
    lcd.DrawLine({0.0f, 240.0f}, {799.0f, 240.0f}, stroke, white);
    lcd.DrawLine({400.0f, 0.0f}, {400.0f, 479.0f}, stroke, white);

    // Draw a green quadratic bezier curve (parabaola)
    SolidBrush green({0x00, 0xFF, 0x00});
    lcd.DrawCurve({0.0f, 0.0f}, {400.0f, 800.0f}, {799.0f, 0.0f}, stroke, green);

    // run forever
    while(true);

    return 0;
}
```



Example – Rectangles

```
#include <MoaTouch.h>

using namespace MoaTouch;

int main()
{
    // all lines 1px thick
    Stroke stroke(1.0f);

    // rectangle with sharp corners
    SolidBrush cyan({0x00, 0xFF, 0xFF});
    lcd.DrawRectangle({{50.0f, 100.0f}, 150.0f, 200.0f}, stroke, cyan);

    // rounded rectangle with 20 pixel uniform corner radii.
    SolidBrush yellow({0xFF, 0xFF, 0x00});
    lcd.DrawRectangle({{100.0f, 150.0f}, 150.0f, 200.0f}, 20.0f, stroke, yellow);

    // filled rounded rectangle with 20px uniform corner radii
    SolidBrush green({0x00, 0xFF, 0x00});
    lcd.FillRectangle({{300.0f, 50.0f}, 100.0f, 100.0f}, 20.0f, green);

    // filled rectangle with sharp corners
    SolidBrush transFuchsia({0xFF, 0x00, 0xFF, 0x55});
    lcd.FillRectangle({{350.0f, 100.0f}, 100.0f, 100.0f}, transFuchsia);

    // run forever
    while(true);

    return 0;
}
```

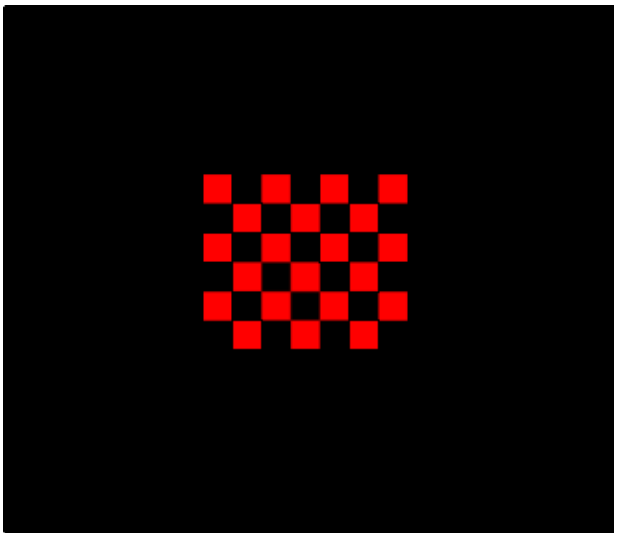


Example – Plotting Pixels

This example draws a small checkerboard pattern in the upper left quadrant of the screen.

```
#include <vector>
#include <MoaTouch.h>

using namespace std;
using namespace MoaTouch;

int main()
{
    // specifiy colors
    Color red(0xFF, 0x00, 0x00);
    Color black(0x00, 0x00, 0x00);

    // create a patter for odd and even rows
    vector<Color> oddRow = { red, black, red, black, red, black, red };
    vector<Color> evenRow = { black, red, black, red, black, red, black };

    // display the pattern
    for(int16_t i = 0; i < 6; i += 2)
    {
        lcd.SetPixels({100, 100 + i}, oddRow);
        lcd.SetPixels({100, 100 + i + 1}, evenRow);
    }

    // run forever
    while(true);

    return 0;
}
```

Zoomed in 12x:



`LinearGradientBrush` Class

This class is used with the various drawing features of the MoaTouch to display shapes in a 2-color linear gradient. This class implements the Brush Class.

Constructors

`LinearGradientBrush();`

Instantiates a new uninitialized instance of this class.

`LinearGradientBrush(const GradientStop& start, const GradientStop& end);`

Instantiates an instance of this class with the given start and end `GradientStop`s.

`LinearGradientBrush(const LinearGradientBrush& source);`

Copy constructor

`LinearGradientBrush& operator=(const LinearGradientBrush& source);`

Copy assignment operator

Methods

`MoaTouch::GradientStop GetStart() const;`

Returns a copy of the start `GradientStop`

`void SetStart(const GradientStop& value);`

Sets the start `GradientStop` to the given value

`MoaTouch::GradientStop GetEnd() const;`

Returns a copy of the stop `GradientStop`

`void SetEnd(const GradientStop& value);`

Sets the stop `GradientStop` to the given value

`MoaTouch::BrushType GetType() const;`

Returns `BrushType::LinearGradient` to identify this brush as a linear gradient brush.

Operators

`LinearGradientBrush& operator=(const LinearGradientBrush& source);`

Copy assignment operator

Example

```
#include <MoaTouch.h>

int main()
{
    // start off with a blank screen
    lcd.SetBacklightBrightness(255);
    lcd.EnableFlush();
    lcd.Clear();
    while(lcd.HasChanges());

    // Create a gray-to-blue gradient brush
    Color gray(0xC0, 0xC0, 0xC0);
    Color blue(0x33, 0x66, 0x99);
    GradientStop start({400.0f, 200.0f}, gray);
    GradientStop end({400.0f, 280.0f}, blue);
    LinearGradientBrush brush(start, end);

    // Display a rounded rectangle with the brush
    lcd.FillRectangle({{300.0f, 200.0f}, 200.0f, 80.0f}, 20.0f, brush);
```

```
    // run forever
    while(true);

    return 0;
}
```

`LineCap` Enumeration

This enumeration is used with the Stroke Class to define the appearance of the stroke's end caps.

Items

`Butt`

Line ends abruptly right a the coordinate of the end point.

`Round`

End point is given a rounded cap that may extend beyond the coordinate of the end point.

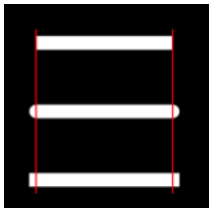
`Square`

End point is given a square cap that may extend beyond the coordinate of the end point

Example

This example shows the appearance of each kind of end cap.

```
#include <MoaTouch.h>

using namespace MoaTouch;

int main()
{
    lcd.SetBacklightBrightness(255);            // start off with a blank screen
    lcd.EnableFlush();
    lcd.Clear();
    while(lcd.HasChanges());

    // draw 10-pixel wide horizontal line with Butt cap
    SolidBrush white({0xFF, 0xFF, 0xFF});
    Stroke horizontalStroke(10.0f, LineCap::Butt);
    lcd.DrawLine({100.0f, 100.0f}, {200.0f, 100.0f}, horizontalStroke, white);

    // draw 10-pixel wide horizontal line with Round cap
    horizontalStroke.SetCap(LineCap::Round);
    lcd.DrawLine({100.0f, 150.0f}, {200.0f, 150.0f}, horizontalStroke, white);

    // draw 10-pixel wide horizontal line with Square cap
    horizontalStroke.SetCap(LineCap::Square);
    lcd.DrawLine({100.0f, 200.0f}, {200.0f, 200.0f}, horizontalStroke, white);

    // draw 1-pixel wide red vertical lines showing the line's end point coordinates
    SolidBrush red({0xFF, 0x00, 0x00});
    Stroke verticalStroke(1.0f);
    lcd.DrawLine({100.0f, 90.0f}, {100.0f, 210.0f}, verticalStroke, red);
    lcd.DrawLine({200.0f, 90.0f}, {200.0f, 210.0f}, verticalStroke, red);

    // run forever
    while(true);

    return 0;
}
```



`LineJoin` Enumeration

This enumeration is used with the Stroke Class to define the appearance of two adjoining lines in a Path.

Items

Miter
Line joint is drawn with a sharp corner.

Round
Line joint is drawn with a rounded corner.

Bevel
Line joint is drawn with a beveled corner.

Example

This example shows the appearance of each kind of end cap.

```
#include <MoaTouch.h>

using namespace MoaTouch;

int main()
{
    // start off with a blank screen
    lcd.SetBacklightBrightness(255);
    lcd.EnableFlush();
    lcd.Clear();
    while(lcd.HasChanges());

    // Draw line with a rounded joint
    SolidBrush white({0xFF, 0xFF, 0xFF});
    Stroke stroke(10.0f, LineCap::Butt, LineJoin::Round);
    Path path;
    path.MoveTo({100.0f, 100.0f});
    path.LineTo({150.0f, 50.0f});
    path.LineTo({200.0f, 100.0f});
    lcd.DrawPath(path, stroke, white);

    // Draw line with a mitered joint
    stroke.SetJoin(LineJoin::Miter);
    path.Clear();
    path.MoveTo({250.0f, 100.0f});
    path.LineTo({300.0f, 50.0f});
    path.LineTo({350.0f, 100.0f});
    lcd.DrawPath(path, stroke, white);

    // Draw line with a beveled joint
    stroke.SetJoin(LineJoin::Bevel);
    path.Clear();
    path.MoveTo({400.0f, 100.0f});
    path.LineTo({450.0f, 50.0f});
    path.LineTo({500.0f, 100.0f});
    lcd.DrawPath(path, stroke, white);

    // run forever
    while(true);

    return 0;
}
```

`MemoryStream` Class

This class can be used to provide stream semantics to a memory buffer. It can be useful for string fonts, images, or other user content in memory for use by some of the MoaTouch's drawing features. It inherits from the FiniteStreamReader Class.

Constructors

```
MemoryStream(uint8_t* buffer, size_t size);
```

Create a memory stream around the given buffer

```
MemoryStream(const MemoryStream& source);
```

Copy constructor

Methods

```
virtual size_t GetCurrentPosition() = 0;
```

Reports the current position in the stream.

```
virtual size_t GetSize() const = 0;
```

Reports the total number of bytes in the stream.

```
size_t Read(void* buffer, size_t numOfBytes);
```

Read `numOfBytes` from the current position in the stream and store the data read in `buffer`. This method will also advance the current position in the stream by `numOfBytes`.

```
virtual void Seek(size_t position) = 0;
```

Moves to the given position in the stream.

Example

See Fonts and Text for a demonstration of this class.

`ModbusRTUMaster` Class

This class allows the MoaTouch to function as a Modbus RTU master for controlling other PLCs over one of the MoaTouch's serial ports.

`ModbusRTUMaster::Status` Enum

An enumeration for identifying the current status of this class.

Items

`ReadyForRequest = 0`

Ready to begin a new query

`WaitingForResponse = 1`

Waiting for the respone from the last query

`Timeout = 2`

Did not receive a response within the given timeout

`ResponseReceived = 3`

Response successfully received

`CRCError = 4`

Response contained a CRC error

Constructors

```
ModbusRTUMaster(SerialPort& serialPort, uint32_t timeoutInMilliseconds);
```

Instantiates a new instance of this class over the given `serialPort`. The given `serialPort` should be opened and configured separately using the SerialPort Class's features. `timeoutInMilliseconds` specifies how long Modbus queries should wait for a response.

Methods

```
void ReadCoilStatus(uint8_t slaveAddress, uint16_t startAddress, uint16_t count);
```

Performs a Modbus Read Coils (function code 1) query. This function will throw a `std::runtime_error` if the status is `Status::WaitingForResponse`. It is recommended to verify the status with the `GetStatus` method before calling this method.

- `slaveAddress` – The Modbus slave address of the device to query
- `startAddress` – The Modbus start address of the first coil to read
- `count` - The number of coils to read.

```
void ReadInputStatus(uint8_t slaveAddress, uint16_t startAddress, uint16_t count);
```

Performs a Modbus Read Discreet Input (function code 2) query. This function will throw a `std::runtime_error` if the status is `Status::WaitingForResponse`. It is recommended to verify the status with the `GetStatus` method before calling this method.

- `slaveAddress` – The Modbus slave address of the device to query
- `startAddress` – The Modbus start address of the first input to read
- `count` - The number of inputs to read.

```
void ReadHoldingRegisters(uint8_t slaveAddress, uint16_t startAddress, uint16_t count);
```

Performs a Read Holding Register (function code 3) query. This function will throw a `std::runtime_error` if the status is `Status::WaitingForResponse`. It is recommended to verify the status with the `GetStatus` method before calling this method.

- `slaveAddress` – The Modbus slave address of the device to query
- `startAddress` – The Modbus start address of the first register to read
- `count` - The number of registers to read.

```
void ReadInputRegisters(uint8_t slaveAddress, uint16_t startAddress, uint16_t count);
```

Performs a Read Input Register (function code 4) query. This function will throw a `std::runtime_error` if the status is `Status::WaitingForResponse`. It is recommended to verify the status with the `GetStatus` method before calling this method.

- `slaveAddress` – The Modbus slave address of the device to query
- `startAddress` – The Modbus start address of the first register to read
- `count` - The number of registers to read.

```
void ForceSingleCoil(uint8_t slaveAddress, uint16_t startAddress, bool value);
```

Performs a Write Single Coil (function code 5) query. This function will throw a `std::runtime_error` if the status is `Status::WaitingForResponse`. It is recommended to verify the status with the `GetStatus` method before calling this method.

- `slaveAddress` – The Modbus slave address of the device to query
- `startAddress` – The Modbus address of the first coil to write
- `value` – The value to write to the coil

```
void ForceMultipleCoils(uint8_t slaveAddress, uint16_t startAddress, const std::vector<bool>&
values);
```

Performs a Write Mutiple Coils (function code 15) query. This function will throw a `std::runtime_error` if the status is `Status::WaitingForResponse`. It is recommended to verify the status with the `GetStatus` method before calling this method.

- `slaveAddress` – The Modbus slave address of the device to query
- `startAddress` – The Modbus start address of the first coil to write
- `values` – The collection of sequential values to write

```
void PresetSingleRegister(uint8_t slaveAddress, uint16_t startAddress, uint16_t value);
```

Performs a Write Single Register (function code 6) query. This function will throw a `std::runtime_error` if the status is `Status::WaitingForResponse`. It is recommended to verify the status with the `GetStatus` method before calling this method.

- `slaveAddress` – The Modbus slave address of the device to query
- `startAddress` – The Modbus address of the register to write
- `value` – The value to write to the register

```
void PresetMultipleRegisters(uint8_t slaveAddress, uint16_t startAddress, const
std::vector<uint16_t>& values);
```

Performs a Write Multiple Registers (function code 16) query. This function will throw a `std::runtime_error` if the status is `Status::WaitingForResponse`. It is recommended to verify the status with the `GetStatus` method before calling this method.

- `slaveAddress` – The Modbus slave address of the device to query
- `startAddress` – The Modbus address of the first register to write
- `value` – The collection of sequential values to write

```
Status GetStatus();
```

Gets the current operating status of this instance.

```
Status WaitForResponse();
```

Wait until a response to the current query has been received or an error. The status ending the wait is returned.

```
void GetResponse(std::vector<uint16_t>& values);
```

Gets the response from a register query (function codes 3, 4, 6, 16). This method will throw a `std::runtime_error` if the status is not `Status::ResponseReceived`. It is recommended to verify the status with the `GetStatus` method before calling this method. The result is written to `values`.

```
void GetResponse(std::vector<bool>& values);
```

Gets the response from a boolean query (function codes 1, 2, 5, 15). This method will throw a `std::runtime_error` if

the status is not `Status::ResponseReceived`. It is recommended to verify the status with the `GetStatus` method before calling this method. The result is written to `values`.

Example

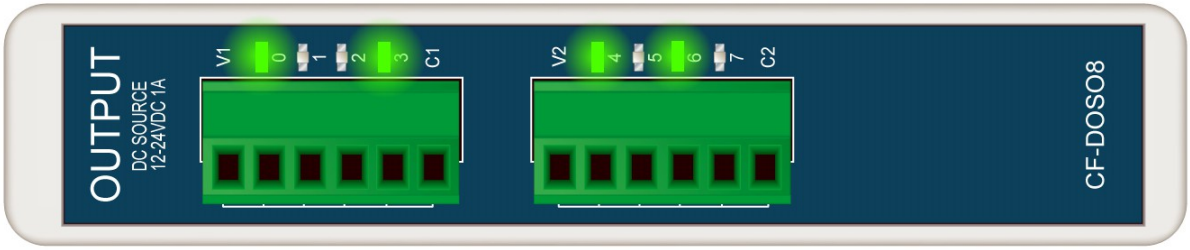
This example illustrates a combination of the UI, Touch, and Modbus features of the MoaTouch. It uses the `ModbusRTUMaster` Class to control Comfile Technology's Moacon MD-DOS08 Digital Output Source module. When the operator touches one of the `LEDButton`s on the screen, the MD-DOSO8's corresponding output state will toggle.

```
#include <MoaTouch.h>

using namespace MoaTouch;

static volatile bool handledTouch;
static Point touchedPoint;

static void OnTouch(Point p)
{
    // Capture the point touched by the operator
    touchedPoint = p;

    // Let UI know that it needs to handle the touch event
    handledTouch = false;
}

static void OnMove(Point p)
{
    // If the operator's touch has not yet been handled by the UI...
    if (!handledTouch)
    {
        // ... use the latest point touched by the operator
        touchedPoint = p;
    }
}

static void OnRelease(Point p)
{
    // Don't have the UI handle any more touch events
    handledTouch = true;
}

// A button that if touched will toggle the state of a ModPort output while displaying the
// output's state
class LEDButton
{
public:
    LEDButton(uint8_t index, ModbusRTUMaster* modbus)
        : m_index(index)
        , m_area({25.0f + (m_index * 100.0f), 50.0f}, 50.0f, 30.0f)
        , m_modbus(modbus)
        , m_state(false)
    {
        // Draw the UI with the ModPort output's initial state
        Render();
        Update();
    }

    // Process the touch event
    bool ProcessTouch(Point& p)
    {
        // Did the operator touch this LED
        bool isTouched = m_area.Intersects(p);
        if (isTouched)
        {
            // Toggle the ModPort output via Modbus
            m_modbus->ForceSingleCoil(1, m_index, !m_state);
            m_modbus->WaitForResponse();
        }

        // Let caller know the this LED handled the touch event
        return isTouched;
    }
```

```
    // Read the status of the ModPort output via Modbus and update the UI
    // if something has changed
    void Update()
    {
        // Read the status of the ModPort output via Modbus
        m_modbus->ReadCoilStatus(1, m_index, 1);
        ModbusRTUMaster::Status status = m_modbus->WaitForResponse();
        if (status == ModbusRTUMaster::Status::ResponseReceived)
        {
            // Get the Modbus response
            vector<bool> response;
            m_modbus->GetResponse(response);

            // if the state has changed...
            if (response[0] != m_state)
            {
                // remember the new state
                m_state = response[0];

                // update the UI
                Render();
            }
        }
    }

private:
    uint8_t          m_index;       // index (0~7) for this LED
    Area             m_area;        // The area of the screen where this LED is displayed
    ModbusRTUMaster* m_modbus;      // pointer to the Modbus RTU Master
    bool             m_state;       // remember the current state so we know when something
                                    // has changed

    // Draw the LED on the screen
    void Render()
    {
        SolidBrush onOffBrush(m_state
            ? Color(0x00, 0xFF, 0x00)           //  Bright green if the LED is ON
            : Color(0x00, 0x50, 0x00));         //  Dark green if the LED is Off

        lcd.FillRectangle(m_area, onOffBrush);  //  Draw the green part

        // Draw a white border around the LED
        SolidBrush white({0xFF, 0xFF, 0xFF});
        Stroke stroke(2.0, LineCap::Butt, LineJoin::Round);
        lcd.DrawRectangle(m_area, stroke, white);
    }
};

int main()
{
    // Configure callbacks for receiving touch events
    touch.SetOnTouch(OnTouch);
    touch.SetOnMove(OnMove);
    touch.SetOnRelease(OnRelease);

    // Configure the serial port
    serialPorts[1].SetBaudRate(57600);
    serialPorts[1].Open(32, 32);

    // Configure the Modbus RTU Master
    ModbusRTUMaster modbus(serialPorts[1], 100);

    // Create an array of 8 LEDs corresponding the ModPort's 8
    // outputs on the MD-DOSI8
    const uint8_t NUM_OF_LEDS = 8;
    LEDButton leds[NUM_OF_LEDS] =
    {
        LEDButton(0, &modbus),
        LEDButton(1, &modbus),
        LEDButton(2, &modbus),
        LEDButton(3, &modbus),
```

```
        LEDButton(4, &modbus),
        LEDButton(5, &modbus),
        LEDButton(6, &modbus),
        LEDButton(7, &modbus)
    };

    // Run program forever
    while(true)
    {
        // If the operator touched the screen, but our program has not yet
        // handled the touch event.
        if (!handledTouch)
        {
            // Give each LED the chance to process the touch event
            for(uint8_t i = 0; i < NUM_OF_LEDS; i++)
            {
                // a short delay to improve reliability
                Delay_ms(1);

                // Give the LED a chance to process the touch event
                if (leds[i].ProcessTouch(touchedPoint))
                {
                    // If the touch corresponds to this LED, no need to
                    // check the other LEDs.  Just exit.
                    break;
                }
            }

            // Don't process any more touches until the operator's finger
            // is released
            handledTouch = true;
        }

        // Give each LED the chance to update its state
        for(uint8_t i = 0; i < NUM_OF_LEDS; i++)
        {
            // Update the UI with the latest state
            leds[i].Update();
        }
    }

    return 0;
}
```

MoaTouch



Moacon

NVRam Class

The NVRam class provides read and write access to the MoaTouch's non-volatile memory, so data can be retained on the MoaTouch between power cycles. It is implemented as a singleton class whose instance can be accessed via the `nvram` identifier.

Methods

`size_t GetCapacityInBytes() const;`

Gets the capacity of the non-volatile memory in bytes

`void Read(uint8_t address, void* buffer, size_t numOfBytes);`

Read a given number of bytes, `numOfBytes`, from the specified non-volatile memory address, `address`, and copy the values into the given memory buffer, `buffer`. This method will throw a `std::range_error` if `address` and `numOfBytes` implies and address not within the non-volatile memory's capacity.

`void Write(uint8_t address, const void* buffer, size_t numOfBytes);`

Read a given number of bytes, `numOfBytes`, from the given memory buffer, `buffer`, and write the values to the specified non-volatile memory address, `address`. This method will throw a `std::range_error` if address and `numOfBytes` implies an address not within the non-volatile memory's capacity.

Example

This example will write to every location in the non-volatile memory and then read back and verify the value.

```
#include <MoaTouch>

using namespace MoaTouch;

int main()
{
    // write values one byte at a time to the non-volatile memory
    console.PrintLine("Writing Values");
    for(size_t i = 0; i < nvram.GetCapacityInBytes(); i++)
    {
        uint8_t value = i % 256;
        nvram.Write(i, &value, 1);
    }

    // read and verify the values from the non-volatile memory one byte at a time.
    console.PrintLine("Reading Values");
    for(size_t i = 0; i < nvram.GetCapacityInBytes(); i++)
    {
        uint8_t value;
        uint8_t expectedValue = i % 256;

        // read the value from non-volatile memory
        nvram.Read(i, &value, 1);

        // verify the value
        if (value != expectedValue)
        {
            console.PrintLine("Error: %d != %d", value, expectedValue);
            break;
        }
    }

    // run forever
    console.PrintLine("Finished");
    while(true);

    return 0;
}
```

Console window output:

```
Writing Values
Reading Values
```

```
Finished
```

`Path` Class

This class is used to draw arbitrary shapes.

Factory methods

The following methods are static methods that construct and return a new instance of a path. They can be called with the `Path::FunctionName` syntax.

```
static Path Line(const Point& p0, const Point& p1);
```
Create a line from point `p0` to point `p1`

```
static Path Curve(const Point& from, const Point& control, const Point& to);
```
Create a bezier curve from point `from` to point `to` with control point `control`

```
static Path Curve(const Point& from, const Point& control0, const Point& control1, const Point& to);
```
Create a bezier curve from point `from` to point `to` with control points `control0` and `control1`

```
static Path Arc(const Point& center, float horizontalRadius, float verticalRadius, float startAngle,
float sweep);
```
Create an elliptical arc starting at `startAngle`, traveling `sweep` degrees.

```
static Path Rectangle(const Area& area, float cornerRadius = 0);
```
Create a rectangle or rounded rectangle encompassing the given `area` with uniform corner radii `cornerRadius`.

```
static Path Rectangle(
    const Area& area,
    float topLeftConerRadius,
    float topRightConerRadius,
    float bottomLeftConerRadius,
    float bottomRightConerRadius);
```
Create a rounded rectangle encompassing the given `area` with nonuniform corner radii.

```
static Path Circle(const Point& center, float radius);
```
Create a circle with the given `center` and `radius`.

```
static Path Ellipse(const Point& center, float horizontalRadius, float verticalRadius);
```
Create a ellipse

```
static Path Text(
    const Point& point,
    const FiniteStreamReader& font,
    const std::u32string& text,
    float size,
    float angle = 0.0f);
```
Create a path from the given `text` using the given `font`

```
static Path Text(
    const Point& point,
    const FiniteStreamReader& font,
    const std::string& text,
    float size,
    float angle = 0.0f);
```
Create a path from the given `text` using the given `font`

Constructors

```
Path();
```
Create an empty path

`Path(const Path& source);`
Copy constructor

Methods

`void Clear();`
Clear the path removing all vertices.

`void MoveTo(const Point& to);`
Move to point `to` to begin drawing. See Example - Pie Section, Example - Curve for a demonstration.

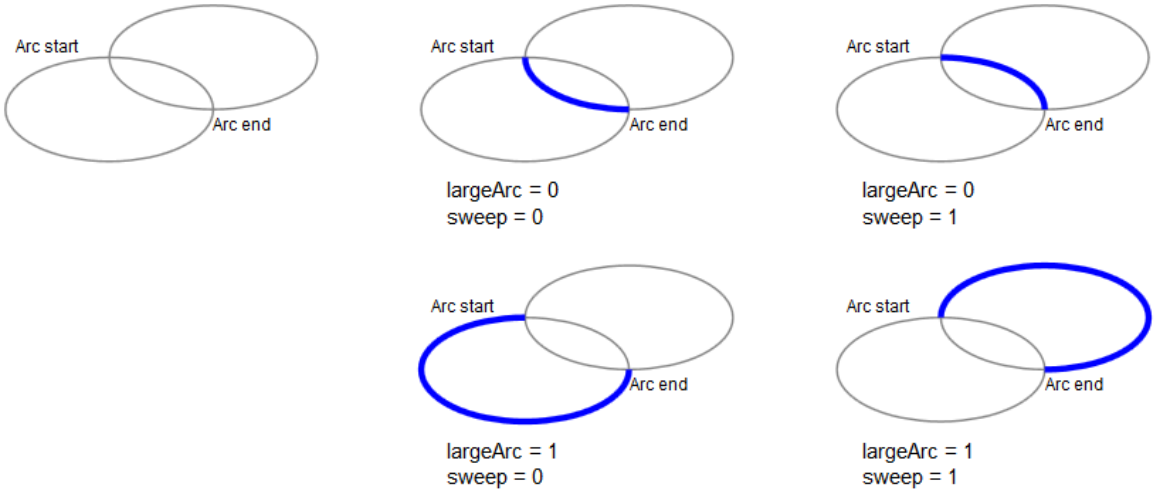
`void LineTo(const Point& to);`
Draw a line form the current point to the point `to`. This method can be used to make polylines. See Example - Pie Section for a demonstration.

`void CurveTo(const Point& control, const Point& to);`
Draw a bezier curve from the current point to point `to` with control point `control`.

`void CurveTo(const Point& control0, const Point& control1, const Point& to);`
Draw a bezier curve from the current point to point `to` with control points `control0` and `control1`. See Example - Curve for a demonstration.

`void ArcTo(float horizontalRadius, float verticalRadius, float rotationAngle, const bool largeArc, const bool sweep, const Point& to);`
Draw an elliptical arc from the current point to point `to`. See Example - Pie Section for a demonstration.

The following illustrates the affect of the `largeArc` and `sweep` arguments.



`void Close();`
Close the shape. See Example - Pie Section for a demonstration.

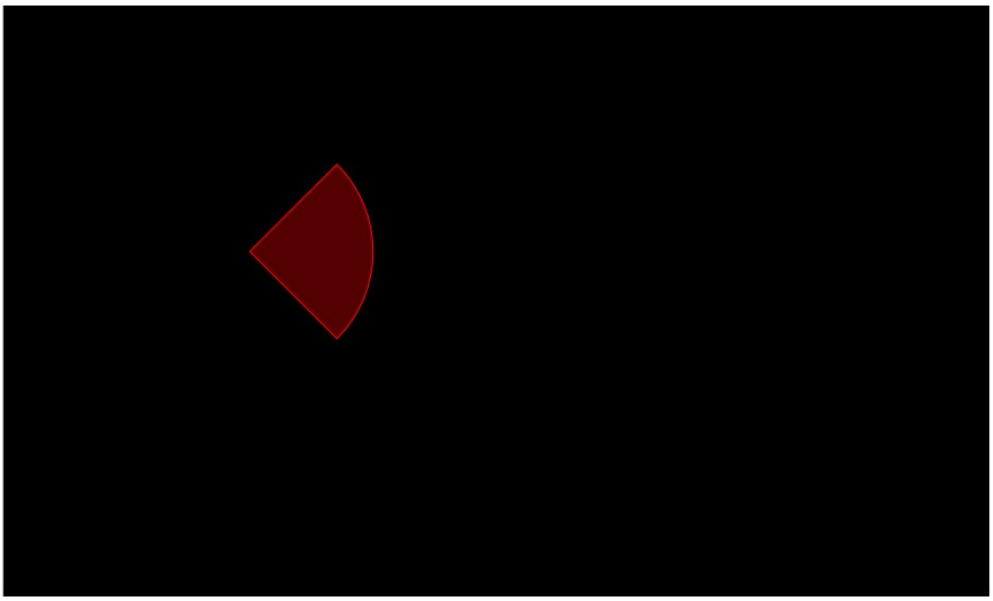
Operators

`Path& operator=(const Path& source);`
Copy assignment operator

Example – Pie Section

```
#include <MoaTouch.h>
using namespace MoaTouch;
```

```
int main()
{
    // Create a section of a pie
    Path path;
    path.MoveTo({200.0f, 200.0f});
    path.LineTo({270.7f, 129.29f});
    path.ArcTo(100.0f, 100.0f, 0.0f, false, true, {270.7, 270.7});
    path.Close();

    // Fill the section with dark red
    SolidBrush darkRed({0x54, 0x00, 0x00});
    lcd.FillPath(path, darkRed);

    // Outline the section in bright red
    SolidBrush brightRed({0xFF, 0x00, 0x00});
    Stroke stroke(1.0f);
    lcd.DrawPath(path, stroke, brightRed);

    while(true);

    return 0;
}
```



Example – Curve

```
#include <MoaTouch.h>
using namespace MoaTouch;

int main()
{
    // Create a curve
    Path path;
    path.MoveTo({200.0f, 200.0f});
    path.CurveTo({300.0f, 300.0f}, {300.0f, 100.0f}, {400.0f, 200.0f});

    // Draw the curve
    SolidBrush cyan({0x00, 0xFF, 0xFF});
    Stroke stroke(5.0f);
    lcd.DrawPath(path, stroke, cyan);

    while(true);

    return 0;
}
```

`Parity` Enumeration

This enumeration is used with the SerialPort Class to specify it's parity setting. Note that if using `Parity::Odd` or `Parity::Even`, the `SerialPort`'s DataBits must be `DataBits::Bits9` to accommodate the parity bit.

Items

`None = 0`

No parity

`Odd = 1`

Odd parity

`Even = 2`

Even parity

Example

Please see the SerialPort Class Example for a demonstration of this enumeration.

`PixelArea` Class

A 2-dimensional rectangular area in whole pixel units. It is identical to the Area Class, but has units of whole pixels intended for use the raster graphics where exactly pixel alignment is needed.

Constructors

```
Area();
```
Creates a area at 0,0 with a width and height of 0

```
Area(int16_t x0, int16_t y0, int16_t x1, int16_t y1);
```
Creates a rectangular area from four coordinates identifying the points at opposite corners on a diagonal.

```
Area(Point p0, Point p1);
```
Creates a rectangular area from two points at opposite corners on a diagonal.

```
Area(Point topLeft, uint16_t width, uint16_t height);
```
Creates a rectangular area with the top-left corner at `topLeft`, with a width of `width` and a height of `height`.

Methods

```
int16_t GetBottom() const;
Gets the bottom-most y-coordinate of this area

Point GetBottomLeft() const;
```
Gets the point that is the bottom-left corner of this area

```
Point GetBottomRight() const;
Gets the point that is the bottom-right corner of this area

uint16_t GetHeight() const;
Gets the height of this area

Area GetIntersection(const Area& area) const;
Returns an area that is the intersection of the given area and this area

int16_t GetLeft() const;
```
Gets the left-most x-coordinate of this area

```
int16_t GetRight() const;
```
Gets the right-most x-coordinate of this area

```
int16_t GetTop() const;
```
Gets the top-most y-coordinate of this area

```
Point GetTopLeft() const;
```
Gets the point that is the top-left corner of this area

```
Point GetTopRight() const;
```
Gets the point that is the top-right corner of this area

```
uint16_t GetWidth() const;
```
Gets the width of this area

```
bool Intersects(const Point& point) const;
```
Returns true if the given point intersects this area

```
bool Intersects(const Area& area) const;
```
Returns true if the given area and this area intersect one another

```
bool IsOffBottomBound(float value) const;
```
Returns true if the given coordinate lies below this area

```
bool IsOffBottomRightBound(const Point& point) const;
```
Returns true if the given point is either to the right of this area, below this area, or both.

```
bool IsOffLeftBound(float value) const;
```

Returns true if the given coordinate lies to the left of this area

```
bool IsOffRightBound(int16_t value) const;
```
Returns true if the given coordinate lies to the right of this area

```
bool IsOffTopBound(int16_t value) const;
```
Returns true if the given coordinate lies above this area

```
bool IsOffTopLeftBound(const Point& point) const;
```
Returns true if the given point is either to the left of this area, above this area, or both.

```
bool IsWithinHorizontalBounds(int16_t value) const;
```
Returns ture if the given coordinate is greater than or equal to the left coordinate of this area, and less than or equal to the right coordinate of this area.

```
bool IsWithinVerticalBounds(int16_t value) const;
```
Returns true if the given coordinate is greater than or equal to the top coordinate of this area, and less than or equal to the bottom coordinate of this area.

```
void SetHeight(uint16_t value);
```
Sets the height of this area

```
void SetLeft(int16_t x);
```
Sets the left-most x coordinate of this area. Setting this value does not change the width or height of this area. It effectively moves the area along the x-axis

```
void SetTop(int16_t y);
```
Sets the top-most y coordinate of this area. Setting this value does not change the width or height of this area. It effectively moves the area along the y-axis

```
void SetTopLeft(const Point value);
```
Sets the top-left corner of this area. Setting this value does not change the width or height of this area. It effectively shifts the area to a new location.

```
void SetWidth(uint16_t value);
```
Sets the width of this area

`PixelPoint` Class

Represents a point in a 2-dimensional plane in whole pixel units. It is identical to the Point Class, but has units of whole pixels intended for use the raster graphics where exactly pixel alignment is needed.

Constructors

```
Point(int16_t x = 0.0f, int16_t y = 0.0f);
```

Creates a new point at `x,y`

```
Point(const Point& source);
```

Copy constructor

Fields

```
int16_t  x
```

This point's x coordinate

```
int16_t y
```

This point's y coordinate

Methods

```
static float GetHoirzontalDisplacement(const Point& p0, const Point& p1);
```

Gets the horizontal displacement between `p0` and `p1`

```
static float GetVerticalDisplacement(const Point& p0, const Point& p1);
```

Gets the vertical displacement between `p0` and `p1`

```
static float GetDisplacement(const Point& p0, const Point& p1);
```

Gets the displacement between `p0` and `p1`

```
static float GetAngleRadians(const Point& p0, const Point& p1);
```

Gets the angle from the horizontal in radians for the line connecting `p0` and `p1`

Operators

```
Point& operator=(const Point& source);
```

Copy assignment operator

```
bool operator==(const Point& rhs);
```

Returns true if this point's coordinates are equal to `rhs`'s coordiantes

```
bool operator!=(const Point& rhs);
Returns true if this point's coordinates are not equal to rhs's coordiantes
```

`Point` Class

Represents a point in a 2-dimensional plane.

Constructors

```
Point(float x = 0.0f, float y = 0.0f);
```
Creates a new point at `x`,`y`

```
Point(const Point& source);
```
Copy constructor

Fields

```
float x
```
This point's x coordinate

```
float y
```
This point's y coordinate

Methods

```
static float GetHoirzontalDisplacement(const Point& p0, const Point& p1);
```
Gets the horizontal displacement between `p0` and `p1`

```
static float GetVerticalDisplacement(const Point& p0, const Point& p1);
```
Gets the vertical displacement between `p0` and `p1`

```
static float GetDisplacement(const Point& p0, const Point& p1);
```
Gets the displacement between `p0` and `p1`

```
static float GetAngleRadians(const Point& p0, const Point& p1);
```
Gets the angle from the horizontal in radians for the line connecting `p0` and `p1`

Operators

```
Point& operator=(const Point& source);
```
Copy assignment operator

```
bool operator==(const Point& rhs);
```
Returns true if this point's coordinates are equal to `rhs`'s coordiantes

```
bool operator!=(const Point& rhs);
```
Returns true if this point's coordinates are not equal to `rhs`'s coordiantes

`RadialGradientBrush` Class

This class is used with the various drawing features of the MoaTouch to display shapes in a 2-color radial gradient. This class implements the Brush Class.

Constructors

`RadialGradientBrush();`

Instantiate a new uninitialized instance of this class

`RadialGradientBrush(const GradientStop& center, float radius, const Color& outerColor);`

Instantiates a new instance of this class with the given center GradientStop, radius, and outer Color

`RadialGradientBrush(const RadialGradientBrush& source);`

Copy constructor

Methods

`GradientStop GetCenter() const;`

Returns the center GradientStop

`void SetCenter(const GradientStop& value);`

Sets the center GradientStop

`float GetRadius() const;`

Gets the radius from the center GradientStop to the outer color

`void SetRadius(float value);`

Sets the radius from the center GradientStop to the outer color

`Color GetOuterColor() const;`

Gets the outermost color that the gradient should blend to as it gets further from the center.

`void SetOuterColor(const Color& value);`

Sets the outermost color that the gradient should blend to as it gets further from the center.

`BrushType GetType() const;`

Returns `BrushType::RadialGradient` to identify this brush as a radial gradient

Operators

`RadialGradientBrush& operator=(const RadialGradientBrush& source);`

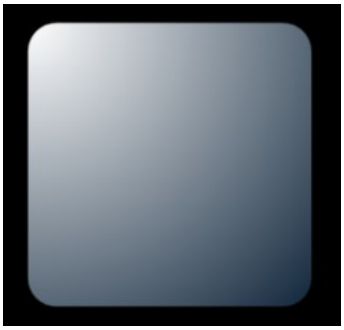
Copy assignment operator

Example

This example will display a rounded-rectangle in a gray-to-blue radial gradient from the top-left corner to the bottom-right corner.

```
#include <MoaTouch.h>

using namespace MoaTouch;

int main()
{
    // start off with a blank screen
    lcd.SetBacklightBrightness(255);
    lcd.EnableFlush();
    lcd.Clear();
    while(lcd.HasChanges());

    // Create a gray-to-blue radial gradient brush
    Color gray(0xC0, 0xC0, 0xC0);
    Color blue(0x33, 0x66, 0x99);
```

```
    GradientStop start({300.0f, 140.0f}, gray);
    RadialGradientBrush brush(start, 280.0f, blue);

    // Display a rounded rectangle with the brush
    lcd.FillRectangle({{300.0f, 140.0f}, 200.0f, 200.0f}, 20.0f, brush);

    // run forever
    while(true);

    return 0;
}
```



RTC Class

This class provides access to the MoaTouch's real-time clock. It is implemented as a singleton class whose instance can be accessed via the `rtc` identifier. See also the DateTime Class.

Methods

```
DateTime GetDateTime();
```

Get the current date and time

```
void SetDateTime(const DateTime& value);
```

Set the current date and time

Examples

See the DateTime Class Example for a demonstration of the class.

`ScreenOrientation` Enumeration

An enumeration for specifying the orientation of the MoaTouch's screen. See the `LCD::SetOrientation` method.

Items

`Rotate0 = 0`

Default - Screen not rotated (800 x 480).

`Rotate90 = 90`

Screen rotated 90 degrees clockwise (480 x 800).

`Rotate180 = 180`

Screen rotated 180 degrees essentially appear upsided-down from the default (800 x 480).

`Rotate270 = 270`

Screen rotated 270 degrees, or 90 degrees counter-clockwise (480 x 800).

`SerialPort` Class

This is the class used for programming the MoaTouch's 4 serial ports. Each port is implemented as a singleton accessible via the `serialPorts` identifier. The MoaTouch has 2 RS-485 ports (`serialports[0~1]`) and 2 RS-232 ports (`serialPorts[2~3]`).

`SerialPort::Channel` Enumeration

An enumeration for identifying an individual serial port channel.

Methods

`Channel GetChannel() const;`
Gets the channel associated with this serial port

`uint32_t GetBaudRate() const;`
Gets the baud rate for this serial port

`void SetBaudRate(const uint32_t value);`
Sets the baud rate for this serial port

`Parity GetParity() const;`
Gets the parity setting for this serial port. See the Parity Enumeration.

`void SetParity(const Parity value);`
Sets the parity setting for this serial port. See the Parity Enumeration.

`DataBits GetDataBits() const;`
Gets the data bits setting for this serial port. See the DataBits Enumeration.

`void SetDataBits(const DataBits value);`
Sets the data bits setting for this serial port. See the DataBits Enumeration.

`StopBits GetStopBits() const;`
Gets the stop bits setting for this serial port. See the StopBits Enumeration.

`void SetStopBits(const StopBits value);`
Sets the stop bits setting for this serial port. See the StopBits Enumeration.

`size_t GetRxCapacity() const;`
Gets the receive buffer capacity for this serial port as set in the Open function

`size_t GetTxCapacity() const;`
Gets the transmit buffer capacity for this serial port as set in the Open function

`void Open(size_t txCapacity, size_t rxCapacity);`
Opens this serial port, allocating memory for the transmit and receive buffers.

`void Close();`
Close this serial port, deallocating the transmit and receive buffers.

`bool IsOpen() const;`
Returns true if the port is opened

`size_t GetRxCount() const;`
Gets the number of bytes currently in the receive buffer

`size_t GetTxCount() const;`
Gets the number of bytes currently in the transmit buffer

`size_t Write(const void* buffer, const size_t size);`
Writes `size` bytes from `buffer` to the transmit buffer. Returns the number of bytes written to the transmit buffer

`size_t WriteByte(uint8_t value);`
Writes a single byte to the transmit buffer. Returns the number of bytes written to the transmit buffer.

`size_t Read(void* buffer, const size_t size);`
Reads `size` bytes from the receive buffer to `buffer`. Returns the number of bytes copied.

`int ReadByte();`
Reads a single byte from the receive buffer. Returns the byte read from the receive buffer, or a negative number if the receive buffer is empty.

`void Print(const char c);`
Writes a single char to the transmit buffer

`void Print(const char* c);`
Writes a null terminated string to the transmit buffer

`void Print(const std::string& s);`
Writes a string to the transmit buffer

`void PrintLine(const char* c);`
Writes a null-terminated string followed by a newline to the transmit buffer

`void PrintLine(const std::string& s);`
Writes a string followed by a newline to the transmit buffer

`std::string GetNewLine() const;`
Gets a string to use for newlines.

`void SetNewLine(const std::string& value = u8"\r\n");`
Sets the string to use for newlines

`std::function<void()> GetOnDataReceived() const;`
Gets the function that is to be called when data is received from the serial port

`void SetOnDataReceived(std::function<void()> value);`
Sets the function that is to be called when data is received from the serial port

`std::function<void(const SerialError)> GetOnError() const;`
Gets the function that is to be called when a serial port error occurs

`void SetOnError(std::function<void(const SerialError)> value);`
Sets the function that is to be called when a serial port error occurs

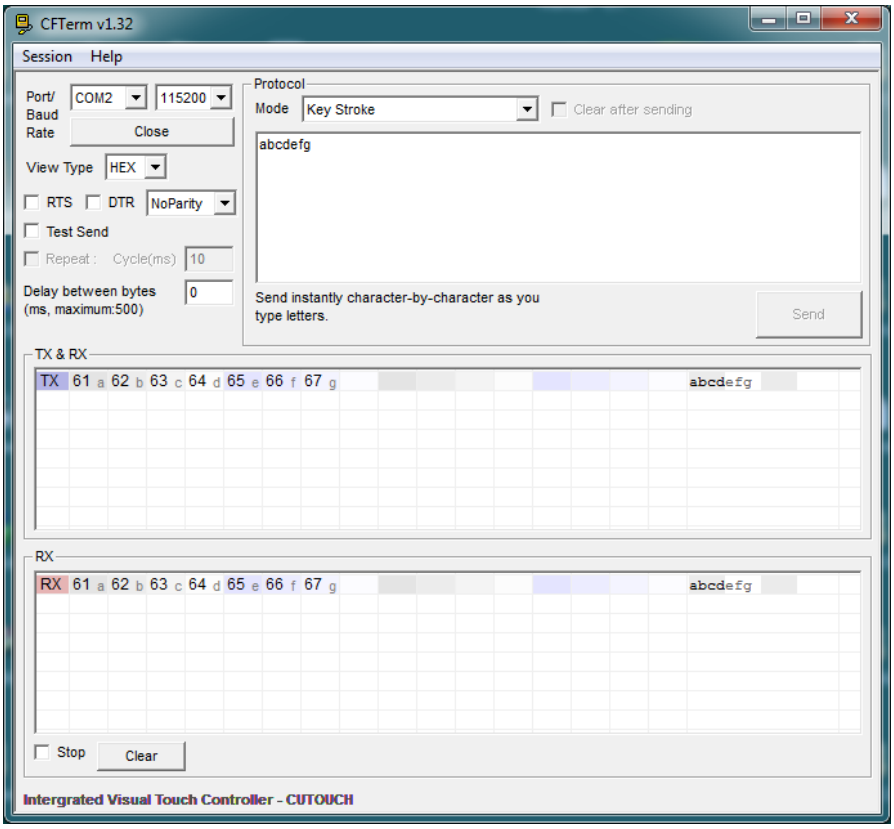
Example

This example echos data received on serial port channel 2 back to the host.

```
int main()
{
    // user serial port channel 2
    SerialPort& serialPort = serialPorts[2];

    // configure the serial port and open it
    serialPort.SetBaudRate(115200);
    serialPort.SetParity(Parity::None);
    serialPort.SetStopBits(StopBits::Bits1);
```

```
    serialPort.SetDataBits(DataBits::Bits8);
    serialPort.Open(32, 32);

    // run forever
    while(true)
    {
        // if data has been received
        if (serialPort.GetRxCount() > 0)
        {
            char buffer[32];

            // read the data received
            size_t bytesRead = serialPort.Read(buffer, 32);

            // echo the data received
            serialPort.Write(buffer, bytesRead);
        }
    }

    return 0;
}
```

Results using Comfile Technology's CFTerm.



SDCard Class

This class provides access to the MoaTouch's micro SC Card. It is implemented as a singleton class whose instance can be accessed via the `sdCard` identifier.

The MoaTouch does not support C file I/O with `fopen`, `fread`, `fwrite`, nor does it support C++ IO streams. Please use this class along with the `File` and `FileInfo Class` for all file I/O.

The MoaTouch only supports 8.3 file names (8 character name, 3 character extension). Due to this standard file names are always displayed in upper-case characters even if lower-case characters are specified.

Methods

```
void Close(File* file);
void Close(File& file);
```
Closes the given open file. Throws a `std::runtime_error` on failure.

```
void Create(const std::string& path);
```
Creates a new file at the given path. If the path ends in a '/' character, a folder will be created. Throws a `std::runtime_error` on failure.

```
bool Exists(const std::string& path);
```
Returns true if a file or folder at the given path exists.

```
void GetChildren(const std::string& path, std::vector<FileInfo>& info);
```
Gets children of the given folder. Throws a `std::runtime_error` on failure.

```
FileInfo GetInfo(const std::string& path);
```
Gets the information on the given file or folder. Throws a `std::runtime_error` on failure.

```
std::function<void()> GetOnEjected() const;
```
Gets the function to be called when an SD Card is ejected

```
std::function<void(const std::exception&)> GetOnError() const;
```
Gets the function to be called if there is an error while accessing the SD Card

```
std::function<void()> GetOnInserted() const;
```
Gets the function to be called when an SD Card is inserted

```
bool IsInserted() const;
```
Returns true if an SD Card is inserted

```
bool IsMounted() const;
```
Returns true if the file system is mounted

```
void Mount();
```
Mount the file system

```
void Move(const std::string& existingPath, const std::string& newPath);
```
Moves a file from `existingPath` to `newPath`. Throws a `std::runtime_error` on failure.

```
File* Open(const std::string& path);
```
Opens a file on the SD Card. Throws a `std::runtime_error` on failure.

```
void Remove(const std::string& path);
```
Removes the file or folder at the given path. Throws a `std::runtime_error` on failure.

```
void Delete(const std::string& path);
```
Deletes the file or folder at the given path (same as `Remove`). Throws a `std::runtime_error` on failure.

`void SetOnInserted(std::function<void()> value);`
Sets the function to be called when an SD Card is inserted

`void SetOnEjected(std::function<void()> value);`
Sets the function to be called when an SD Card is ejected

`void SetOnError(std::function<void(const std::exception&)> value);`
Sets the function to be called if there is an error while accessing the SD Card

`void Unmount();`
Unmount the file system

Example

This example will iterate through the file system and display the contents of each folder in a hierarchy.

```
#include <MoaTouch.h>
#include <string>
#include <vector>

using namespace std;
using namespace MoaTouch;

// display a message when the SD Card is inserted
static void OnInserted()
{
    console.PrintLine("SD Card Inserted");
}

// display a messsage when the SD Card is ejected
static void OnEjected()
{
    console.PrintLine("SD Card Ejected");
}

// Recursive function to print the contents of a folder
static void PrintFolderContents(const string& path, const string& tab)
{
    // Get all the children in folder path
    vector<FileInfo> children;
    sdCard.GetChildren(path, children);

    // iterate through all of the children
    for(FileInfo c : children)
    {
        // Print the file/folder name
        console.Print(tab);
        console.PrintLine(c.GetName());

        // If a folder, print its contents
        if (c.IsFolder())
        {
            PrintFolderContents(path + "/" + c.GetName(), tab + "  ");
        }
    }
}

int main()
{
    sdCard.SetOnInserted(OnInserted);   // display a message when card is inserted
    sdCard.SetOnEjected(OnEjected);     // display a message when card is ejected

    while(!sdCard.IsInserted());        // wait for card to be inserted

    sdCard.Mount();                     // Mount the filesystem

    PrintFolderContents("/", "");       // Print the contents of the filesystem starting at
                                        // root
```

```
    sdCard.Unmount();                   // umount the filesystem

    while(true);                        // run forever

    return 0;
}
```

Console window output:

```
SD Card Inserted
LEVEL1
  FILE3.TXT
  FILE1.TXT
  FILE2.TXT
  LEVEL2
    FILE3.TXT
    FILE1.TXT
    FILE2.TXT
FILE1.TXT
FILE2.TXT
FILE3.TXT
SD Card Ejected
```

`Socket` Class

An individual TCP/IP socket for performing communication over the MoaTouch's Ethernet port. The MoaTouch has 4 sockets.

`Socket::Number` Enumeration

An enumeration for identifying an individual serial port channel.

`Socket::Status` Enumeration

An enumeration of specific connection states for TCP/IP communication

Items

```
Closed      = 0x00
ARP         = 0x01
Init        = 0x13
Listening   = 0x14
SynSent     = 0x15
SynReceived = 0x16
Established = 0x17
FinWait     = 0x18
Closing     = 0x1A
TimeWait    = 0x1B
CloseWait   = 0x1C
LastAck     = 0x1D
UDP         = 0x22
IPRaw       = 0x32
MACRaw      = 0x42
PPOE        = 0x5F
```

Methods

```
Number GetNumber() const;
```
Gets the socket's number for identification

```
void Open(uint16_t port);
```
Opens this socket using the given TCP/IP port

```
void Close();
```
Close this socket

```
void Listen();
```
Listen for an incoming connection on this socket. This socket should be opened first

```
void Connect(uint8_t oct3, uint8_t oct2, uint8_t oct1, uint8_t oct0, uint16_t port);
```
Establish a connection with to a the given ip address (individual octects) on the specified port

```
void Connect(uint32_t ipAdddress, uint16_t port);
```
Establish a connection with to a the given ip address (32-bit unsigned integer) on the specified port

```
void Disconnect();
```
Disconnect any established connections

```
size_t GetRxCount() const;
```
Gets the number of bytes received on this socket

```
Status GetStatus() const;
```
Gets the current connection state of this socket

`size_t Read(void* buffer, size_t numOfBytes);`

Read `numOfBytes` bytes from this socket's receive buffer and copy the data to buffer. Returns the number of bytes actually read.

`int ReadByte();`

Reads a single byte from the receive buffer. Returns the byte read from the receive buffer, or a negative number if the receive buffer is empty.

`size_t Write(const void* buffer, size_t numOfBytes);`

Copy `numOfBytes` bytes from `buffer` to this socket's transmit buffer. Returns the number of bytes actually written.

`size_t WriteByte(uint8_t value);`

Writes a single byte to the transmit buffer. Returns the number of bytes actually written.

Example

The following example is a simple web server that prints "Hello from the MoaTouch" to a visiting web browser.

```
#include <MoaTouch.h>

using namespace MoaTouch;

int main()
{
    // Configure the Ethernet interface
    ethernet.SetHardwareAddress(0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF);
    ethernet.SetIPAddress(192, 168, 0, 252);
    ethernet.SetSubnetMask(255, 255, 255, 0);
    ethernet.SetGateway(192, 168, 0, 2);

    // Use Socket 0
    Socket& socket = ethernet.GetSocket(Socket::Number::S0);

    // The string to send to the client
    string hello = "Hello from the MoaTouch";

    // Run forever
    while(true)
    {
        // Get the socket's connection status
        Socket::Status status = socket.GetStatus();

        switch(status)
        {
        // If closed, open
        case Socket::Status::Closed:
            socket.Open(8080);
            break;

            // If Initialized, listen for an incoming connection
        case Socket::Status::Init:
            socket.Listen();
            break;

        // When connection is establisheded, send response and disconnect
        case Socket::Status::Established:
            socket.PrintLine("HTTP/1.0 200 OK");
            socket.PrintLine("Content-type: text");
            socket.PrintLine("Content-Length: " + ToString(hello.length()));
            socket.PrintLine("");
            socket.Print(hello);
            socket.Disconnect();
            break;

        default:
            break;
        }
    }
```

```
    return 0;
}
```

<See screenshot on next page>

`SolidBrush` Class

This class is used with the various drawing features of the MoaTouch to display shapes in a solid color. This class implements the Brush Class.

Constructors

`SolidBrush(const Color& color);`

Create a new instance of this class specifying the color

`SolidBrush(const SolidBrush& source);`

Copy constructor

Methods

`Color GetColor() const;`

Gets the color currently used for this brush

`void SetColor(const Color& value);`

Sets the color to use for this brush

`BrushType GetType() const;`

Returns the `BrushType::Solid` to identify this brush as a solid color brush

Operators

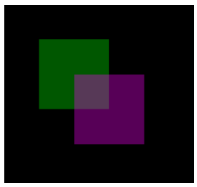
`SolidBrush& operator=(const SolidBrush& source);`

Copy assignment operator

Example

This example will display a partially transparent fuchsia filled rectangle on an opaque green filled rectangle.

```
#include <MoaTouch.h>
using namespace MoaTouch;

int main()
{
    // start off with a blank screen
    lcd.SetBacklightBrightness(255);
    lcd.EnableFlush();
    lcd.Clear();
    while(lcd.HasChanges());

    // display an opaque, green rectangle
    SolidBrush opaqueGreen({0x00, 0xFF, 0x00});
    lcd.FillRectangle({{50.0f, 50.0f}, 100.0f, 100.0f}, opaqueGreen);

    // display a partially transparent fuchsia rectangle
    SolidBrush partiallyTransparentFuchsia({0xFF, 0x00, 0xFF, 0xA9});
    lcd.FillRectangle({{100.0f, 100.0f}, 100.0f, 100.0f}, partiallyTransparentFuchsia);

    while(true);     // run forever

    return 0;
}
```



`StopBits` Enumeration

This enumeration is used with the SerialPort Class to specify the number of stop bits in the `SerialPort`'s settings.

Items

`Bits0_5`
0.5 bits

`Bits1_0`
1.0 bit

`Bits1_5`
1.5 bits

`Bits2_0`
2.0 bits

Example

Please see the SerialPort Class Example for a demonstration of this enumeration.

`Stopwatch` Class

This class can be used to measure time with microsecond precision. It can also be used to insert time delays in a program. It is implemented as a counter which rolls over every 50 seconds, so it cannot be used to measure time beyond that. To measure longer times use the RTC Class.

Constructors

```
Stopwatch();
```
Default constructor

```
Stopwatch(const Stopwatch& source);
```
Copy constructor

Methods

```
static void DelaySeconds(float seconds);
MoaTouch::Delay_s(float seconds);
```
Delay for a given number of seconds.

```
static void DelayMilliseconds(float milliseconds);
MoaTouch::Delay_ms(float milliseconds);
```
Delay for a given number of milliseconds

```
static void DelayMicroseconds(float microseconds);
MoaTouch::Delay_us(float microseconds);
```
Delay for a given number of microseconds

```
float GetElapsedMicroseconds();
```
Get the number of elapsed microseconds.

```
float GetElapsedMilliseconds();
```
Get the number of elapsed milliseconds.

```
float GetElapsedSeconds();
```
Get the number of elapsed microseconds

```
bool IsRunning() const;
```
Returns true if the stopwatch is currently running (counting)

```
void Reset();
```
Resets the counter back to zero without not restarting or stopping the counter.

```
void Restart();
```
Starts the counter after resetting it to 0

```
void Start();
```
Starts or resumes the counter

```
void Stop();
```
Stop or pause the counter

Example

This example will measure the time it takes to draw a rectangle to the LCD's buffer, and display the elapsed time in the console window.

```
#include <MoaTouch.h>

using namespace MoaTouch;
```

```
int main()
{
    lcd.SetBacklightBrightness(255);            // start off with a blank screen
    lcd.EnableFlush();
    lcd.Clear();
    while(lcd.HasChanges());

    SolidBrush cyan({0x00, 0xFF, 0xFF});        // cyan color
    Stroke stroke(1.0f);                        // stroke width of 1.0 pixels

    Stopwatch sw;
    sw.Start();                                 // begin counting

    // draw the rectangle
    lcd.DrawRectangle(50.0f, 100.0f, 150.0f, 200.0f, stroke, cyan);

    sw.Stop();                                  // stop counting

    // display result
    console.PrintLine("%f microseconds", sw.GetElapsedMicroseconds());

    // run forever
    while(true);

    return 0;
}
```

Console window output:

```
6.301357 milliseconds
```

String Functions

The current MoaTouch toolchain does not fully support C99, and therefore does not support the C++11 `std::to_string`, `std::stoi`, `std::stol`, `std:stof`, etc... methods (other C++11 features are supported, however). To compensate for this and to provide a convenient Unicode conversion, the following functions are available as free functions in the `MoaTouch` namespace.

```
std::string ToString(int value);
std::string ToString(long value);
std::string ToString(long long value);
std::string ToString(unsigned value);
std::string ToString(unsigned long value);
std::string ToString(unsigned long long value);
std::string ToString(float value);
std::string ToString(double value);
std::string ToString(long double value);
```

Converts numbers to a decimal string representation. Alternatively, users can always use the C `sprintf` function to convert a number to a string. Using `stringstream` or any of the other C++ io stream features are not recommended as they drastically increase code size.

```
int ToInt(const std::string& s, size_t* pos = 0, int base = 10);
```

Parses a string to an integer. Same as std::stoi.

```
long ToLong(const std::string& s, size_t* pos = 0, int base = 10);
```

Parses a string to a long integer. Same as std::stol.

```
unsigned long ToULong(const std::string& s, size_t* pos = 0, int base = 10);
```

Parses a string to an unsigned long integer. Same as std::stoul.

```
float MoaTouch::ToFloat(const std::string& s, size_t* pos);
```

Parses a string to a single-precision floating point. Same as std::stof.

```
double MoaTouch::ToDouble(const std::string& s, size_t* pos);
```

Parses a string to a double-precision floating point. Same as std::stod.

```
std::u32string ToUTF32(const std::string& s);
```

Converts a UTF-8 string to a UTF-32 string

```
std::string ToUTF8(const std::u32string& s);
```

Converts a UTF-32 string to a UTF-8 string

`Stroke` Class

This class represents the characteristics of the line used to draw the outline of shapes.

Constructors

```
Stroke(float width = 2.0f, LineCap cap = LineCap::Butt, LineJoin join = LineJoin::Miter);
```

Creates a new stroke with the given `width`, `cap`, and `join` characteristics

```
Stroke(const Stroke& source);
```

Copy constructor

Methods

```
float GetWidth() const;
```

Gets the width of the stroke (line width)

```
void SetWidth(float value);
```

Sets the width of the stroke (line width)

```
LineCap GetCap() const;
```

Gets the stroke's line cap (end cap)

```
void SetCap(LineCap value);
```

Sets the stroke's line cap (end cap)

```
LineJoin GetJoin() const;
```

Gets the shape of the line joints for this stroke

```
void SetJoin(LineJoin value);
```

Sets the shape of the line joints for this stroke

Operators

```
Stroke& operator=(const Stroke& source);
```

Copy assignment operator

Example

See the LineCap Enumeration Example and the LineJoin Enumeration Example for a demonstration of this class.

`System` Class

This class provides access to some general features for controlling the MoaTouch. It is implemented as a singleton accessible via the `system` identifier.

Methods

```
void Reset();
```
Perform a software reset.

```
void EnableInterrupts(bool enable = true);
```
Enable interrupts

```
void DisableInterrupts(bool disable = true);
```
Disable interrupts

```
std::function<void(const std::exception&)> GetOnUnhandledException() const;
```
Gets the function to call if an unhandled exception occurs

```
void SetOnUnhandledException(std::function<void(const std::exception&)> value);
```
Sets the function to call if an unhandled exception occurs

```
void AtomicCall(std::function<void(void*)> f, void* args);
```
Runs the given function `f` with arguments `args` at a high priority so it cannot be interrupted.

```
uint32_t GetLibraryMajorVersion() const;
```
Gets the major version number of the MoaTouch library

```
uint32_t GetLibraryMinorVersion() const;
```
Gets the minor version number of the MoaTouch library

```
uint32_t GetHardwareMajorVersion() const;
```
Gets the major version number of the MoaTouch hardware

```
uint32_t GetHardwareMinorVersion() const;
```
Gets the minor version number of the MoaTouch hardware

`Timer` Class

This class provides a periodic timer to interrupt a program a given interval. The timer is implemented as a singleton accessible via the `timer` identifier.

The timer's resolution is 1 microsecond so users can specify interrupt intervals with great precision. However, setting the interval to 1 microsecond will cause an interrupt storm where the program will be spending so much time servicing the interrupt, it won't have the resources to do anything else. Therefore, it is recommended to keep the interval greater than 100 microseconds.

Methods

```
Timer::Channel GetChannel();
```

Gets the channel identifying this timer instance.

```
uint16_t GetInterval() const;
```

Gets the time between ticks in microseconds.

```
void SetInterval(uint32_t value);
```

Set the time between ticks in microseconds

```
void EnableInterrupt(bool enable = true);
```

Enable the timer's interrupt.

```
void DisableInterrupt(bool disable = true);
```

Disables the timer's interrupt.

```
std::function<void()> GetOnTick() const;
```

Gets the function to call if when the timer's interval is reached.

```
void SetOnTick(std::function<void()> value);
```

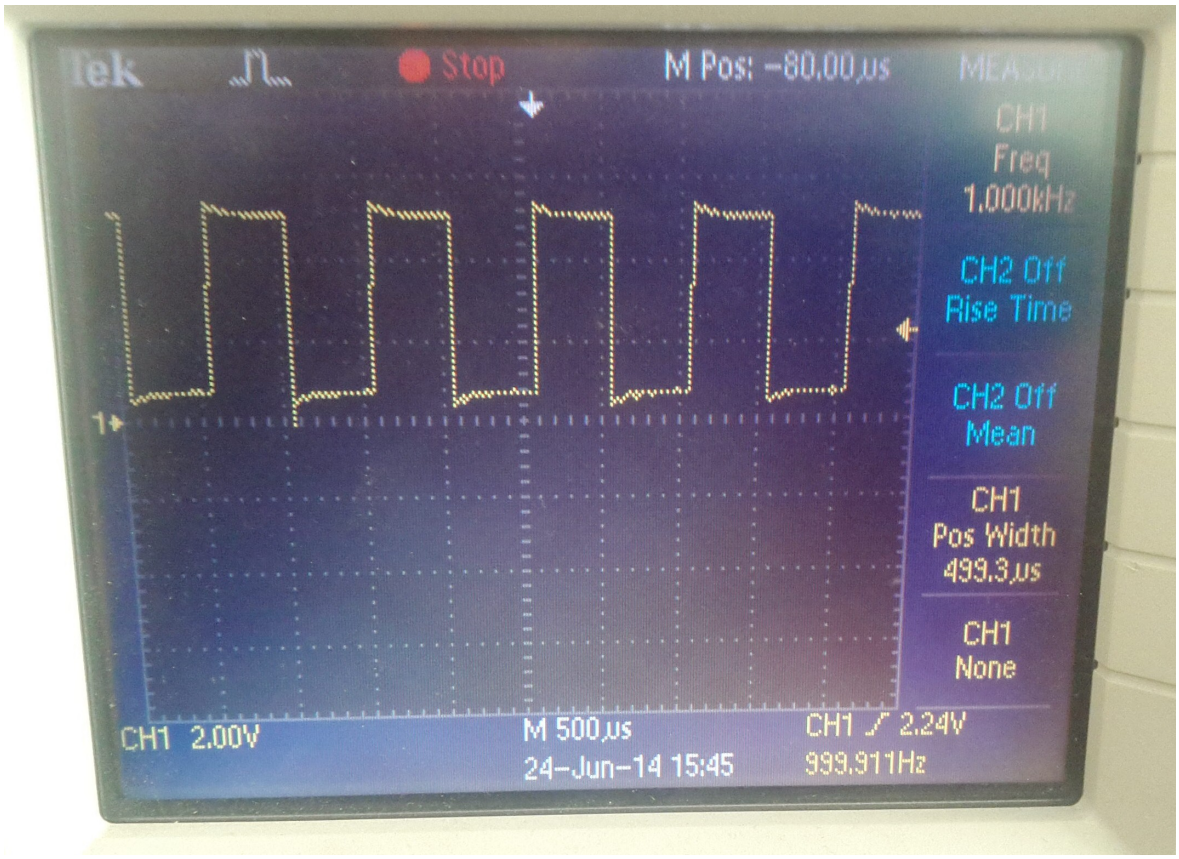
Sets the function to call if when the timer's interval is reached.

Example

```
#include <MoaTouch.h>

using namespace MoaTouch;

void OnTimerTick()
{
    digitalOutputs[0].Toggle();    // Toggle Digital Output 0 when the timer ticks
}

int main()
{
    timer.SetInterval(1000);       // Interrupt every 1 millisecond
    timer.SetOnTick(OnTimerTick);  // Call OnTimerTick on every interrupt
    timer.EnableInterrupt();       // Begin interrupting on every tick

    // run forever
    while(true);

    return 0;
}
```

<See Digital Output 0's waveform on the next page>

The following shows the output of Digital Input 0 on an oscilloscope.



`Touch` Class

This class provides access to the MoaTouch's touch screen. It is implemented as a singleton class whose instance can be accessed via the `touch` identifier.

Methods

```
void Calibrate(std::function<void(Point point)> drawPoint);
```

Calibrates the touch screen. `drawPoint` is a user supplied callback function for displaying the point for the user to touch on the screen.

```
std::function<void(Point point)> GetOnMove() const;
```

Gets the function to call when the user moves (drags) while touching

```
std::function<void(Point point)> GetOnRelease() const;
```

Gets the function to call when a touch is released

```
std::function<void(Point point)> GetOnTouch() const;
```

Gets the function to call when the user first touches the screen.

```
void SetOnRelease(std::function<void(Point point)> onRelease);
```

Sets the function to call when a touch is released. `point` is the last point on the screen that the user touched.

```
void SetOnMove(std::function<void(Point point)> onMove);
```

Sets the function to call when the user moves (drags) while touching. `point` is the point on the screen the user's touch moved to.

```
void SetOnTouch(std::function<void(Point point)> onTouch);
```

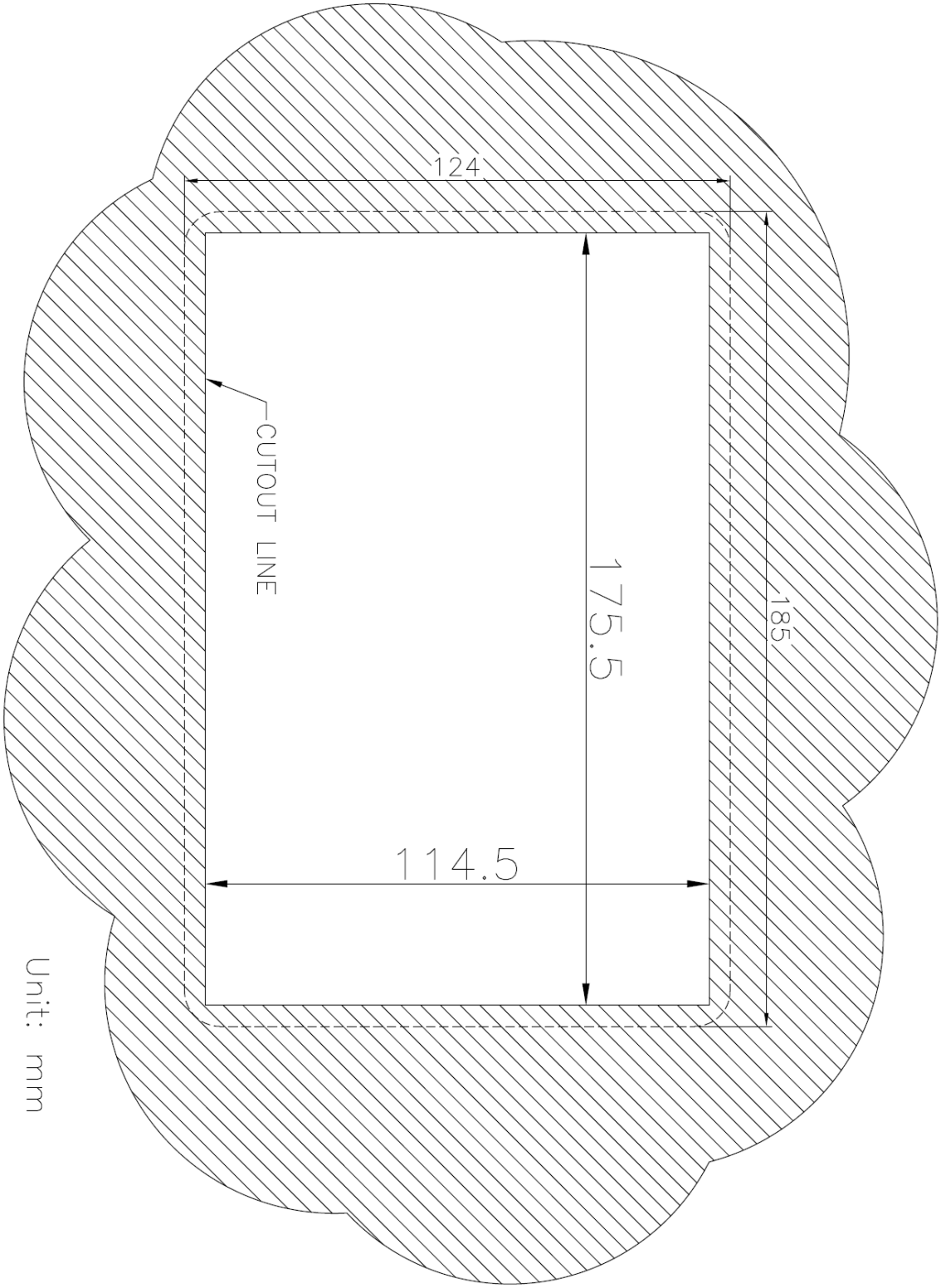
Sets the function to call when the user first touches the screen. `point` is the point on the screen where the user touched.

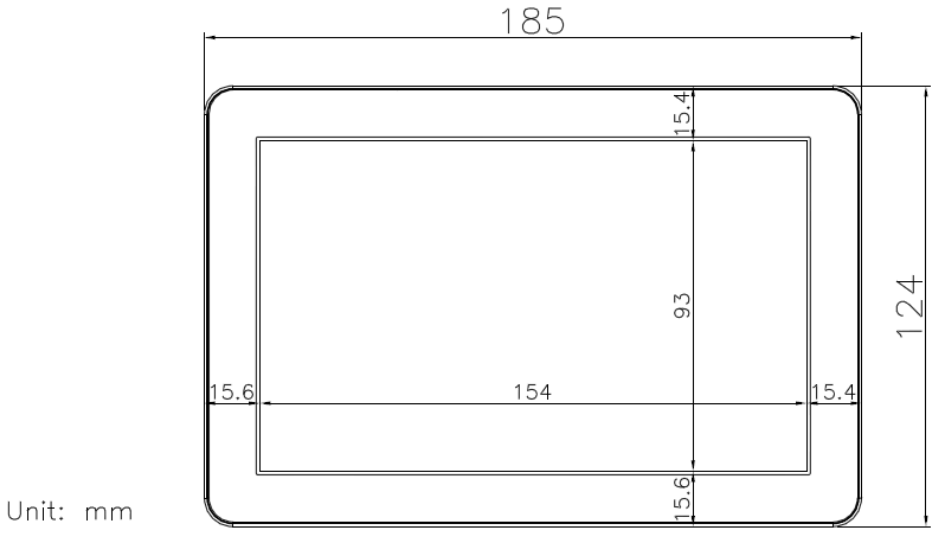
Examples

Please see Calibrating the Touch Screen for an example demonstrating how to calibrate the touch screen with the `Calibrate` method.

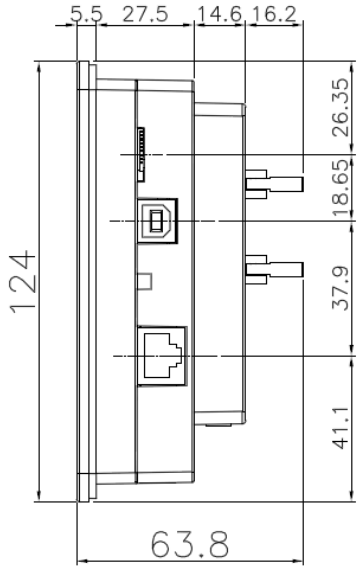
Please see the Buzzer Class Example for a demonstration of responding to the touch, move, and release events.

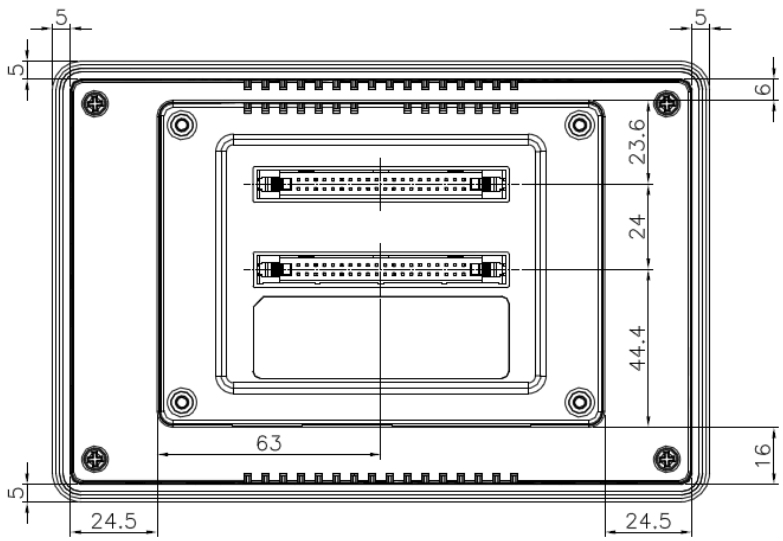
Dimensions

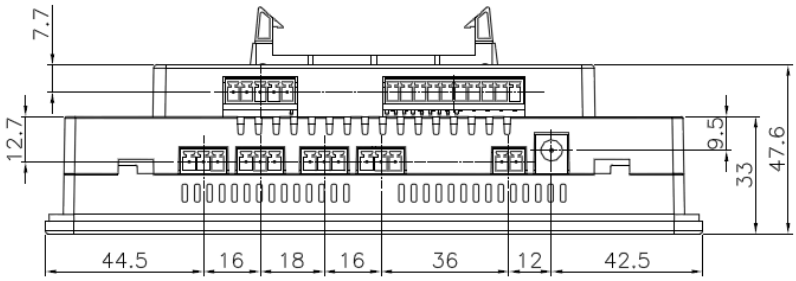


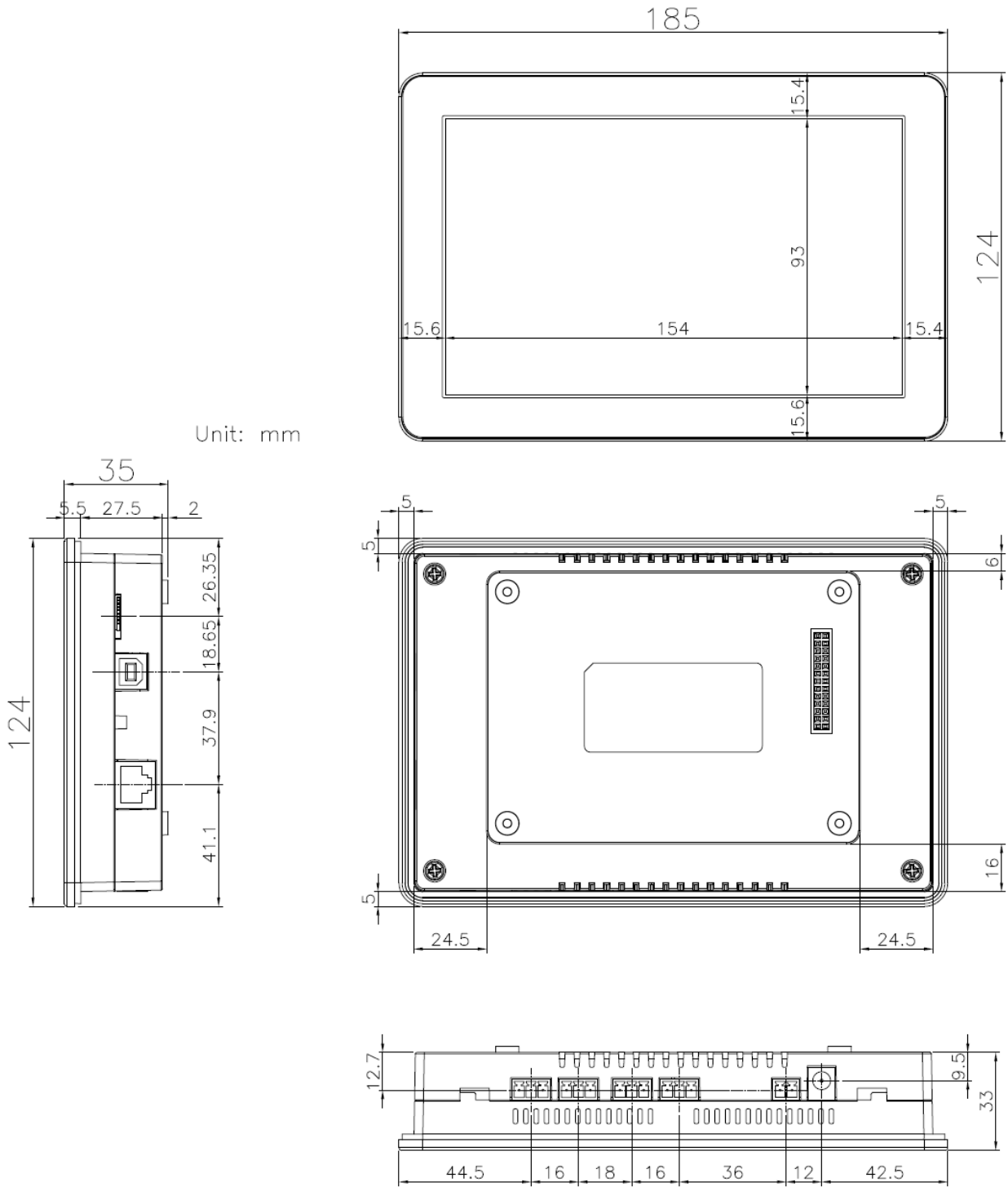

185
15.4
93
124
15.6
154
15.4
15.6

Unit: mm


5.5
27.5
14.6
16.2
26.35
18.65
37.9
41.1
124
63.8


5
5
5
6
23.6
24
44.4
63
16
24.5
24.5


7.7
12.7
9.5
33
47.6
44.5
16
18
16
36
12
42.5


185
15.4
93
124
15.6
154
15.4
15.6
Unit: mm
35
5.5
27.5
2
26.35
18.65
37.9
41.1
124
5
5
5
5
6
16
24.5
24.5
12.7
9.5
33
44.5
16
18
16
36
12
42.5

Attribution

- Portions of the software make use of Anti-Grain Geometry - Version 2.4 Copyright (C) 2002-2004 Maxim Shemanarev (McSeem) http://www.antigrain.com/license/index.html
- Portions of this software are copyright © 2012 The FreeType Project (www.freetype.org). All rights reserved.
- Portions of this software make use of libpng 1.2.50 http://www.libpng.org/pub/png/src/libpng-LICENSE.txt
- Portions of this software make use of zlib 1.2.7 http://zlib.net/zlib_license.html
- Portions of the software are copyright © 2003 Bitstream, Inc. Bitstream Vera is a trademark of Bitstream, Inc. http://dejavu-fonts.org/wiki/License